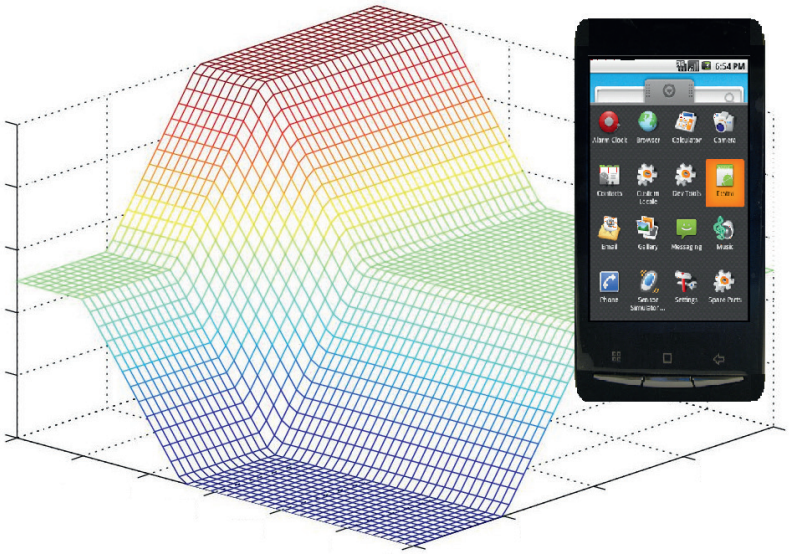
DOCTORAL THESIS

Situation Awareness in Pervasive Computing Systems: Reasoning, Verification, Prediction



Andrey Boytsov

LULEÅ UNIVERSITY OF TECHNOLOGY

Copyright Notices

Notice 1

Notice 2

Situation Awareness in Pervasive Computing Systems: Reasoning, Verification, Prediction

Mr Andrey Boytsov

Submitted in partial fulfillment of the requirements for the
Doctor of Philosophy (Dual Award) (Lulea University of Technology)

Department of Computer Science, Electrical and Space Engineering
Luleå University of Technology
SE-971 87 Luleå, Sweden

Caulfield School of IT
Monash University (MU)
VIC 3145, Australia.

October 2012

Supervisors

Professor Arkady Zaslavsky, Ph.D.,
Luleå University of Technology and CSIRO
Docent Kåre Synnes, Ph.D.,
Luleå University of Technology
Assoc. Professor Shonali Krishnaswamy, Ph.D.,
Monash University

Declaration

I declare that the thesis contains no materials that have been accepted for the award of any degree or diploma at any university unless regulated by LTU requirements towards Licenciate degree. I declare that, to the best of my knowledge, the thesis contains no materials previously published or written by any other person except where due reference is made in the text.

Si 

Date: October, 30, 2012

Luleå University of Technology,
Luleå, Sweden.

Printed by Universitetstryckeriet, Luleå 2013

ISSN: 1402-1544
ISBN 978-91-7439-639-3 (print)
ISBN 978-91-7439-640-9 (pdf)

Luleå 2013

www.ltu.se

To my family.

Abstract

The paradigm of pervasive computing aims to integrate the computing technologies in a graceful and transparent manner, and make computing solutions available anywhere and at any time. Different aspects of pervasive computing, like smart homes, smart offices, social networks, micromarketing applications, PDAs are becoming a part of everyday life.

Context can be defined as information that can be of possible interest to the system. Context often includes location, time, activity, surroundings among other attributes. One of the core features of pervasive computing systems is context awareness – the ability to use context to improve the performance of the system and make its behavior more intelligent.

Situation awareness is related to context awareness, and can be viewed as the highest level of context generalization. Situations allow eliciting the most important information from context. For example, situations can correspond to locations of interest, actions and locomotion of the user, environmental conditions.

The thesis proposes, justifies and evaluates situation modeling methods that allow covering broad range of real-life situations of interest and reasoning efficiently about situation relationships. The thesis also addresses and contributes to learning the situations out of unlabeled data. One of the main challenges of that approach is understanding the meaning of a newly acquired situation and assigning a proper label to it. This thesis proposes methods to infer situations from unlabeled context history, as well as methods to assign proper labels to the inferred situations. This thesis proposes and evaluates novel methods for formal verification of context and situation models. Proposed formal verification significantly reduces misinterpretation and misdetection errors in situation aware systems. The proper use of verification can help building more reliable and dependable pervasive computing systems and avoid the inconsistent context awareness and situation awareness results. The thesis also proposes a set of context prediction and situation prediction methods on top of enhanced situation awareness mechanisms. Being aware of the future situations enables a pervasive computing system to choose the most efficient strategies to achieve its stated objectives and therefore a timely response to the upcoming situation can be provided. In order to become efficient, situation prediction should be complemented with proper acting on prediction results, i.e. proactive adaptation. This thesis proposes proactive adaptation solutions based on reinforcement learning techniques, in contrast to the majority of current approaches that solve situation prediction and proactive adaptation problems sequentially. This thesis contributes to situation awareness field and addresses multiple aspects of situation awareness.

The proposed methods were implemented as parts of ECSTRA (Enhanced Context Spaces Theory-based Reasoning Architecture) framework. ECSTRA framework has proven to be efficient and feasible solution for real life pervasive computing systems.

Table of Contents

Table of Figures

Table of Tables

Preface

Since I got my first computer (Intel 80286 with 1MB RAM and EGA display) at the age of 11, I knew that after school I am going to continue my education in the area of computer science and technology. The interest of exploration lead my first efforts in computer science during the school years, starting with extracurricular BASIC classes for schoolchildren at the age of 12 and proceeding to enrollment into BSc course in computer science at the age of 17.

I obtained BSc and MSc degrees with distinction in computer science from Saint-Petersburg State Polytechnical University in 2006 and 2008 respectively. By that time I already had some positive experience working as a software developer, but what I really wanted was to become a researcher in that field.

In late 2008 I was offered a position of PhD student in LTU and I pursued that opportunity. Later I also joined Monash University as a double degree student. It was a good chance to make some contribution into an emerging area and become a part of the research community.

I defended my licentiate thesis in June, 2011 and continued towards doctoral thesis. Internship in INRIA in November-December 2011 gave me valuable experience and improved my practical knowledge of pervasive computing area. My PhD studies were positive and valuable experience of how the academic world looks like and how the research is carried out.

Luleå, October 2012
Andrey Boytsov

Publications

This thesis consists of introduction, conclusion and 11 chapters, which comprise the contribution of 11 publications. Thesis also contain appendix, which includes a statement of accomplishment from INRIA.

Publications, included in this thesis:

Paper A (reference [BZ10a], chapter I and chapter VIII). Boytsov, A. and Zaslavsky, A. Context prediction in pervasive computing systems: achievements and challenges. in Burstein, F., Brézillon, P. and Zaslavsky, A. eds. *Supporting real time decision-making: the role of context in decision support on the move*. Springer p. 35-64. 30 p. (Annals of Information Systems; 13), 2010.

Paper B (reference [BZ11a], chapter II). Boytsov, A. and Zaslavsky, A. ECSTRA: distributed context reasoning framework for pervasive computing systems. in Balandin, S., Koucheryavy, Y. and Hu H. eds. *Proceedings of the 11th international conference and 4th international con ference on Smart spaces and next generation wired/wireless networking (NEW2AN'11/ruSMART'11)*, Springer-Verlag, Berlin, Heidelberg, 1-13.

Paper C (reference [BZ11b], chapter III). Boytsov, A. and Zaslavsky, A. From Sensory Data to Situation Awareness: Enhanced Context Spaces Theory Approach, in *Proceedings of IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC)*, 2011, pp.207-214, 12-14 Dec. 2011. doi: 10.1109/DASC.2011.55.

Paper D (reference [BZ12a], chapter IV). Boytsov, A., Zaslavsky, A. and Abdallah, Z. Where Have You Been? Using Location Clustering and Context Awareness to Understand Places of Interest. in Andreev, S., Balandin, S. and Koucheryavy, Y. eds. *Internet of Things, Smart Spaces, and Next Generation Networking*, vol. 7469, Springer Berlin / Heidelberg, 2012, pp. 51–62.

Paper E (reference [KB12], chapter V). Kikhia, B., Boytsov, A., Hallberg, J., ul Hussain Sani, Z., Jonsson, H. and Synnes, K.. Structuring and Presenting Lifelogs based on Location Data. Technical report. 2012. 19p. URL= http://pure.ltu.se/portal/files/40259696/KB12_StructuringPresentingLifelogs_TR.pdf, last accessed October, 30, 2012.[1]

[1] The revised technical report was submitted to Personal and Ubiquitous Computing journal.

Paper F (reference [BZ12b], chapter VI). Boytsov, A. and Zaslavsky, A. Formal verification of context and situation models in pervasive computing. *Pervasive and Mobile Computing*, Volume 9, Issue 1, February 2013, Pages 98-117, ISSN 1574-1192, 10.1016/j.pmcj.2012.03.001.
URL=http://www.sciencedirect.com/science/article/pii/S1574119212000417, last accessed May, 08, 2013.

Paper G (reference [BZ11c], chapter VI). Boytsov, A. and Zaslavsky, A. Formal Verification of the Context Model - Enhanced Context Spaces Theory Approach. Scientific report, 2011, 41 p.
URL=http://pure.ltu.se/portal/files/32810947/BoytsovZaslavsky_Verification_TechReport.pdf, last accessed October, 30, 2012.

Paper H (reference [BZ12c], chapter VII). Boytsov, A. and Zaslavsky, A. Correctness Analysis and Verification of Fuzzy Situations in Situation Aware Pervasive Computing Systems. Scientific report, 2013, 30p.
URL= http://pure.ltu.se/portal/files/42973133/BoytsovZaslavsky_FuzzyVerifReport.pdf, last accessed May, 08, 2013.[2]

Paper I (reference [BZ09], chapter IX). Boytsov, A., Zaslavsky, A. and Synnes, K. Extending Context Spaces Theory by Predicting Run-Time Context, in *Proceedings of the 9th International Conference on Smart Spaces and Next Generation Wired/Wireless Networking and Second Conference on Smart Spaces*. St. Petersburg, Russia: Springer-Verlag, 2009, pp. 8-21.

Paper J (reference [BZ10b], chapter X). Boytsov, A. and Zaslavsky, A. Extending Context Spaces Theory by Proactive Adaptation. in Balandin, S., Dunaytsev, R. and Koucheryavy, Y. eds. *Proceedings of the 10th international conference and 3rd international conference on Smart spaces and next generation wired/wireless networking (NEW2AN'10/ruSMART'10)*, Springer Berlin / Heidelberg, 2010, pp. 1-12.

Paper K (reference [Bo10], chapter X). Boytsov, A. Proactive Adaptation in Pervasive Computing Systems, in *ICPS '10: Proceedings of the 7th international conference on Pervasive services*, Berlin, Germany: ACM, 2010.

Some introductory and concluding sections are partially based on the content of the licentiate thesis:

Licentiate thesis (reference [Bo11]). Boytsov, A. Context Reasoning, Context Prediction and Proactive Adaptation in Pervasive Computing Systems. *Licentiate thesis. Department of Computer Science, Electrical and Space Engineering*, Luleå University of Technology, 2011.
URL=http://pure.ltu.se/portal/files/32946690/Andrey_Boytsov.Komplett.pdf, last accessed October, 30, 2012.

[2] The revised technical report is planned for submission to Personal and Ubiquitous Computing journal.

Acknowledgements

I'd like to thank my supervisor Arkady Zaslavsky for his guidance, support, and discussion and comments on my work. I also want to thank Arkady for the provided opportunities to join LTU and to enroll in a double-degree program with Monash University. And I'd like to thank Arkady for helping me to settle in Luleå and in Melbourne.

I'd also like to thanks my co-supervisors Kåre Synnes and Shonali Krishnaswamy for their guidance, discussion and comments, numerous valuable advices, and constant positive attitude.

I want to thank Christer Åhlund for his assistance and support throughout the project.

I want to thank Josef Hallberg for his assistance and encouragement.

I'd like to thank WATTALYST project for providing the fundings for my research.

I want to thank Miguel Castano, Tatiana Boytsova, Natalia Dudarenko, Johan Carlsson, Campbell Wilson and all others who provided me helpful advices and fruitful discussions. Their valuable input inspired the creativity and allowed the research to progress.

I'd also like to thank my colleagues from Monash University, Melbourne, who promoted the collaboration between Monash University and LTU, and provided me the chance to enroll in a double degree program. In particular, I'd like to thank Shonali Krishnaswamy and Mark Carman from Monash University, who provided valuable inputs about my work and encouraged the research collaboration. I'd also like to thank Zahraa Abdallah for her collaboration efforts.

I'd like to thank Sven Molin for his assistance and for governing LTU-Monash collaboration program.

I want to thank Basel Kikhia for his efforts to establish the collaboration between projects and for his numerous ideas for joint research work. I'd also like to thank all those who contributed to our collaborative research.

I want to thank Yuri Karpov and Department of Distributed Computing and Networking of Saint-Petersburg State Polytechnical University for providing education and constant support throughout Bachelors and Masters Studies.

I'd also like to thank Mikael Larsmark for numerous fixes of my equipment.

I'd like to thanks Johan Borg for providing me soldering lessons.

I want to thank Michele Dominici, Fredrik Weis and INRIA for providing me the internship opportunity, which improved my research.

I'd like to thank all my friends and co-workers, who created warm and friendly social environment.

Once again I'd like to thank Arkady Zaslavsky, Kåre Synnes and LTU for their understanding and acting on compassionate grounds.

I want to thank my family, for always being there for me. Especially I want to thank my wife, Renata Esayan, for her patience, encouragement and constant support.

Luleå, March 2013
Andrey Boytsov

Introduction

Situation Awareness in Pervasive Computing Systems: Definition, Verification and Prediction of Situations.

Situation Awareness in Pervasive Computing. Definition, Verification and Prediction of Situations.[3]

1 Pervasive and Ubiquitous Computing

The first research efforts in the field of ubiquitous computing started in 1988, at Xerox Palo Alto Research Center (PARC) [WG99]. What began as an idea of a "computer wall", emerged into a novel computing paradigm.

The paradigm of desktop computing focuses on the use of personal computers – general purpose information processing devices with high computational power. That paradigm is still in use and it functions well for a wide range of tasks. However, the researchers from PARC identified the following shortcomings of the personal computers: "too complex and hard to use; too demanding of attention; too isolating from other people and activities; and too dominating as it colonized our desktops and our lives."[WG99].

The paradigm of ubiquitous computing aims to address those problems by intertwining the computing technologies with everyday life to the extent when the technologies become indistinguishable from it [We91].

Ubiquitous computing solutions are now becoming an integrated part of the everyday environment. Various implementation of ubiquitous computing paradigm include smart homes (see figure 1), smart offices and other ambient intelligence solutions, wearable computing devices, personal digital assistants, social networks. However, there is still much research to be done in the area.

The concept of pervasive computing is connected to the concept of ubiquitous computing so closely, that those terms are sometimes used interchangeably even in the research community [Po09]. It was noted that "the vision of ubiquitous computing and ubiquitous communication is only possible if pervasive, perfectly interoperable mobile and fixed networks exist…" [IS99].

Although often used as synonyms, the term pervasive computing is often preferred when discussing the integration of computing devices and weaving them into the everyday environment, while the term ubiquitous computing is usually preferred when addressing the interfaces and graceful interaction with the user. Based on the provided definitions, this thesis is mostly focused on pervasive computing challenges, and therefore the term "pervasive computing" is used in most cases.

The paradigm of pervasive computing pursues two main goals:

1. Graceful integration of computing technologies into everyday life.
2. High availability – the computing services should be available everywhere and at any time.

[3] The introduction is partially based on the introductory part of the licentiate thesis [Bo11].

Fig. 1. Smart home environment. (a) Kitchen; (b) Fridge; (c),(d) Control panels. Photos taken at DAI - Labor, TU Berlin, Germany in 2010.

Pervasive computing systems often deal with enormous amounts of information, and tend to utilize the large amount of small, highly specialized and highly heterogeneous devices, and those features make the achievement of those goals especially complicated.

The area of pervasive computing proposes new research tasks for the computer science community, and this thesis contributes to overcoming those challenges.

2 Context, Context Awareness and Situation Awareness

Context is a key characteristic of any pervasive computing system. According to the widely acknowledged definition given by Day and Abowd [DA00], context is "any information that can be used to characterize situation of an entity". In plain words, any piece of information that the system has is a part of the system's context. The aspects of context include, but are not limited to, location, identity, activity, time. In this thesis the terms "context", "context data" and "context information" are used interchangeably.

The system is context aware "if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."[DA00]. In simple words, the definition means that the system is context aware if it can use the context information to its benefit. Although recognized as an interdisciplinary area, context awareness is often associated with pervasive computing. Context awareness is core functionality in pervasive computing, and any pervasive computing system is context aware to some extent.

Figure 2 provides an overview of how the context is processed and how the pervasive computing system actions emerge from context processing efforts. On figure 2 the context processing is viewed from the aspects of algorithms and information flows, and that aspect is in the focus of this thesis. For simplicity the aspects like hardware, physical communications, interaction protocols are intentionally left out from figure 2.

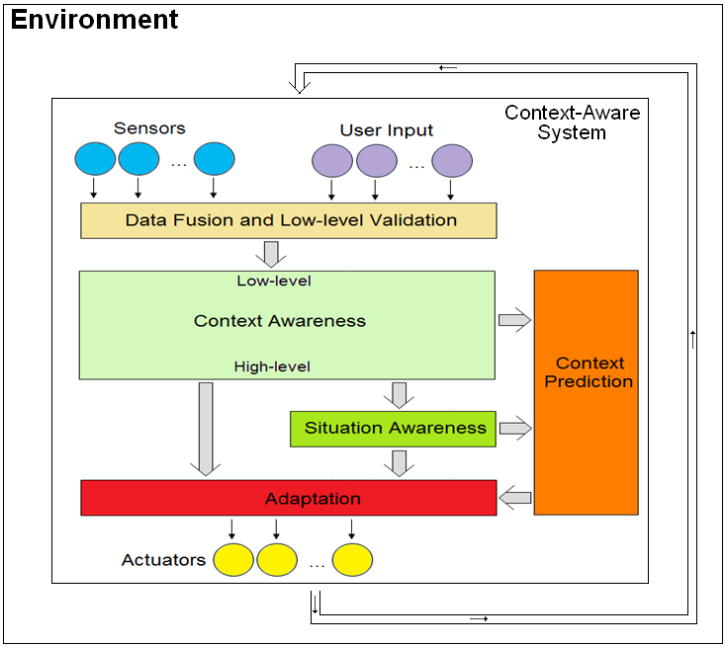


Fig. 2. Context processing

Sensors are the devices that directly measure the environment characteristics (like temperature, light, humidity). Direct user input is provided by such devices as keyboards, touchscreens, and voice recognition solutions. Sensor information and user input are often processed in a similar manner, and in this thesis when talking about sensor information or the input data, both sensory originated information and user input are referred to, unless the distinction is explicitly specified.

After highly heterogeneous input data is delivered, the first processing step is the data fusion and low-level validation of sensor information. Sometimes raw sensor data, collected in a signle vector of values, are already viewed as low-level context.

The distinction between different levels of context grounds in the amount of preprocessing performed upon the collected sensor information. Usually raw or minimally preprocessed sensor data is referred to as low-level context, while the generalized and evaluated information is referred to as high-level context [YD12].

The situation awareness in pervasive computing can be viewed as the highest level of context generalization. Situation awareness aims to formalize and infer real-life situations out of context data. From the perspective of a context aware pervasive computing system, the situation can be identified as "*external semantic interpretation* of sensor data", where the *interpretation* means "situation assigns meaning to sensor data" and *external* means "from the perspective of applications, rather than from sensors" (definitions quoted from the article [YD12]). Therefore, the concept of a situation generalizes the context data and elicits the most important information from it. Properly designed situation awareness

extracts the most relevant information from the context data and provides it in a clear manner. Multiple aspects of situation awareness are the focus of this thesis.

Context prediction aims to predict future context information. It can be done on any level of context processing, starting from low-level context prediction and ending with situation prediction. Different aspects of situation prediction and acting on predicted situation are addressed in the thesis.

Adaptation block defines the response of the pervasive computing system to the provided input, and provides the commands to the actuators. Actuators are the devices that do actions on behalf of pervasive computing environment. For example, a relay that turns switch on or off according to the commands of context aware system is a simple actuator. Or the display that provides the information from context aware system to the user is also an actuator. Or the conditioner that adjusts the temperature on request of smart home environment is also an example of actuator.

The main focus of this thesis is situation awareness. This thesis addresses several important challenges of situation awareness area:

- Properly defining the situations using the expert knowledge.
- Learning the situations from unlabeled context history.
- Ensuring correctness of the obtained situation models.
- Predicting future situations and properly adapting to prediction results

Next section provides more details on what challenges this thesis addresses and what research questions this thesis answers.

3 Research Questions

One of the main goals of situation awareness functionality is sematic interpretation of context information. However, in order to interpret context information, situation aware system needs a mapping between context data and corresponding ongoing situations. For example, if a wearable computing system aims to detect the locomotion of the user, the system needs a model which takes entire set of current sensor readings as input and produces the outputs like “User sits”, “User stands” or “User walks”. Interpretation functionality is the core of situation awareness, and designing that mapping is a challenging and error-prone task. The first research question of this thesis addresses some aspects of that challenge.

Question 1: How to derive a mapping between context information and ongoing situations?

Two possible answers to that question were proposed by research community (for example, see [YD12]).

The first method is to derive the mapping manually using expert knowledge of the subject area. The models based, for example, on ontologies [St09], first order logic [RN09] or fuzzy logic [Pi01] allow the expert to formalize the knowledge of the subject area. At the runtime pervasive computing system can use those formalizations to reason about context and situations. Chapter III of this thesis addresses the challenge of designing situation models in order to achieve ease of development, flexibility and efficient runtime reasoning.

Another option is to learn the mapping from examples. The option of learning the mapping usually refers to supervised learning methods. On the first stage developers observe the situation in practice and create a training set [RN09] – a set of context measurements labeled with an ongoing situation. On the second stage the developers

employ various supervised learning methods to derive the formula, which maps context information to a situation.

This thesis explores a different option of learning the situation, which is less frequently seen in practice – learning situations from unlabeled data. Advantages of that approach include possible learning of new situations at the runtime and becoming aware of the situations that were not considered at the design stage.

One of the main challenges of learning the situations from unlabeled data is the challenge of labeling. For example, cluster of location measurements can correspond to a location of interest to the user, but what kind of location is that? Labeling can be done either manually by the user or automatically. Chapters IV and V of this thesis address both manual and automated labeling.

Both developing and learning the models of situations are complex tasks, and they are prone to various kinds of errors. Those errors can significantly disrupt situation awareness functionality. Research question 2 addresses the methods to detect and fix situation definition errors.

Question 2: How to prove, that the derived mapping is correct?

Consider an example of a wearable computing system, which aims to detect locomotion of the user. Situations like "User sits", "User stands" or "User walks" are represented as formulas, which take sensor readings as inputs and produce probability of a situation as an output. Those formulas can be the subject of expert error, if they are defined by hand. If the formulas are learnt, mistakes can appear, for example, due to overfit or underfit.

Testing the situations is a viable option to detect possible errors, but still sometimes it is not enough. There is no guarantee that a failure scenario will be encountered during testing.

This thesis proposes verification of situation models – a novel method of formally proving situation correctness. Inspired by verification of protocols and software [CG99], verification of situation allows to specify the expected properties of situations and either formally prove that the situations comply with the properties, or derive counterexamples – particular context features that will lead to inconsistent situation awareness.

Situation awareness functionality can be improved by situation prediction. Situation prediction and related aspects constitute research question 3.

Question 3: How to predict future situation and how to act according to prediction results?

Situation prediction is a recognized functionality of pervasive computing systems, and many context prediction systems employ situation prediction. However, there is a lack of general approaches to situation prediction. Many situation prediction solutions were designed to fit their particular tasks and did not mean to be generalized for the entire situation prediction field. Addressing the situation prediction problem in general sense can provide important insights in the area, find the techniques to address common problems of pervasive computing field and derive the methods that are applicable for wide class of situation prediction tasks. This thesis addresses situation prediction challenge and proposes architecture and algorithms to solve situation prediction task.

In order for situation prediction to have any value, pervasive system has to properly act according to prediction results. This task is usually referred to as proactive adaptation task. This thesis addresses the challenge of proactive adaptation and proposes algorithms and architecture to achieve efficient proactive adaptation.

4 Thesis Overview and Roadmap

This thesis consists of 11 chapters, which comprise the contribution of 11 articles. Numeration of figures, formulas and tables is separate for every chapter. The chapters share common references, which are explained in references section at the end of the thesis.

According to LTU dissertation standards thesis includes publications as chapters (except for chapters I and XI). It explains minor formatting differences. The chapters are arranged in the order determined by the research questions. The chosen order is not the chronological order of publications, but it ensures full understanding of how the research proceeds and how the directions of research depend on each other. Chapters II-X are based on my publications with minor modifications. Chapter VI and chapter X contain the results of two merged articles each. The chapters are arranged as follows.

Chapter I sets the necessary background for further work. Chapters I and II address mainly the research question 1, although chapter II is related to all the research questions.

Chapter I contains an overview of situation awareness methods. It discusses the methods to define the situations at the design time or runtime, and elicit the situation out of context data at the runtime. Chapter I describes related work dedicated to defining situations using expert knowledge, learning the situations from labeled data and learning situations from unlabeled data. Chapter I also discusses the challenges of situation awareness and, hence, introduces the background necessary for understanding subsequent chapters and their contribution.

Chapter II proposes ECSTRA (Enhanced Context Spaces Theory-based Reasoning Architecture) – the framework for context awareness and situation awareness. The architecture and implementation of ECSTRA provide solid bases for situation awareness, and gracefully address the problems of context dissemination, multiple agent support and reasoning results sharing. Most of the testing and evaluation, done in this thesis, used either ECSTRA or extensions of it. In collaboration with INRIA, ECSTRA was incorporated in a smart home solution for situation awareness. There ECSTRA has shown practical usefulness, which is certified by INRIA (see appendix).

Chapters III-V address the research question 1.

Chapter III addresses the challenge of defining situations and contains the work to provide extensive situation awareness support for the theory of context spaces. Chapter III proposes enhanced situation models and addresses the aspects of their flexibility, clarity and reasoning complexity.

Chapter IV contributes to one of the least frequently used approaches to situation awareness – learning and labeling situations at the runtime. Chapter IV proposes and tests a novel method to fuse and cluster location information, and then extract relevant places from location data. It views high level location awareness task as situation awareness, and employs situation awareness techniques for location awareness.

Chapter V proposes and proves an alternative approach to the one proposed in chapter IV. Chapter V proposes and evaluates novel location awareness techniques and activity recognition techniques for lifelogging task. Activity recognition and high-level location awareness are viewed as situation awareness tasks. Locations are learned at the runtime, but in contrast with Chapter IV the labels are chosen manually by the user. The application in chapter V aims to make labeling as easy and non-intrusive as possible by providing proper description of locations and activities in terms of location convex hulls and corresponding pictures.

Chapter VI and chapter VII address the research question 2.

Chapter VI proposes and proves the novel technique that allows formal verification of context and situation models in pervasive computing environments. Chapter VI addresses and solves an important problem of context and situation awareness – the plausibility of context model and situation model errors. Once being introduced at the design time, the specification error can lead to inconsistent context awareness and situation awareness results and those results constitute the background of context prediction and proactive adaptation efforts.

Chapter VII continues the research from chapter VI and proposes and proves formal verification method for fuzzy situations. Fuzzy logic is a frequently used technique for situation awareness, and enhancing it with verification capabilities can significantly reduce the number of errors in real life pervasive computing systems.

Chapters VIII-X address the research question 3.

Chapter VIII contains an overview of context prediction and situation prediction in pervasive computing systems. It extensively addresses the massive amount of related work in the area, identifies the features of context prediction task in pervasive computing, proposes the prediction methods comparison criteria and addresses the possible context and situation prediction solution approaches.

Chapter IX addresses the research to apply various context prediction approaches on top of situation awareness capabilities of context spaces theory. The theory of context spaces provides the formalized context awareness and situation awareness approach that can relief the problems of context prediction and proactive adaptation in pervasive computing area. The chapter discusses the possible applications of context prediction techniques both on the level of context models, as well as on the top of situation awareness mechanisms.

Chapter X continues the research direction and introduces proactive adaptation techniques into the context spaces approach. The chapter formally states the task of proactive adaptation, and proves the necessity of an integrated approach to context prediction and proactive adaptation. Chapter X also proposes the possible reinforcement learning mechanisms that can be applied to context spaces theory, and discusses the necessary architectural support for it. As well as in chapter IX, proactive adaptation methods are discussed both at the context model level and on top of situation awareness techniques. Chapter X also proposes CALCHAS (Context Aware Long-term aCt aHead Adaptation System) middleware – an extension of ECSTRA, which aims to improve ECSTRA functionality by context and situation prediction and proper acting according to predicted context.

Chapter XI summarizes the results of the thesis, provides the discussion and possible future work directions.

Figure 3 depicts the roadmap of the thesis, shows the correspondence between the research questions and chapters of the thesis and identifies the connection between the chapters.

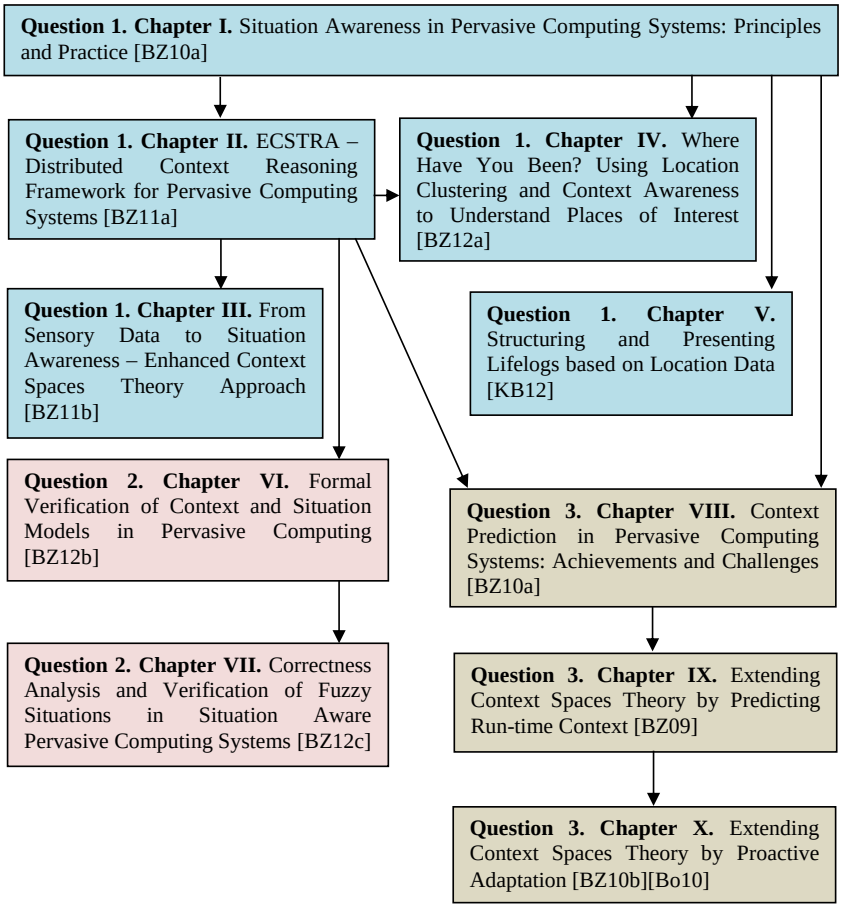


Fig. 3. Thesis roadmap.

Chapter I

Situation Awareness in Pervasive Computing Systems: Principles and Practice

Based on:

1. Boytsov, A. and Zaslavsky, A. Context prediction in pervasive computing systems: achievements and challenges. in Burstein, F., Brézillon, P. and Zaslavsky, A. eds. *Supporting real time decision-making: the role of context in decision support on the move*. Springer p. 35-64. 30 p. (Annals of Information Systems; 13), 2010.[4]

[4] The content of chapter I is largely new. It contains literature review and related work, as well as elements of the article [BZ10a].

Foreword

This chapter contains a survey and classification of existing situation awareness approaches. It overviews related work and discusses current challenges of situation awareness field. Therefore, this chapter provides background information, which is necessary for understanding the contributions of subsequent chapters.

This chapter also starts the answer to the first research question, proposed in this thesis: how to derive a mapping between context information and ongoing situations? Chapter I overviews main groups of methods used to map context features to situations and analyzes main challenges of those methods.

Situation Awareness in Pervasive Computing Systems: Principles and Practice.

1 Context Awareness and Situation Awareness in Pervasive Computing

Context awareness is one of the most important features of pervasive computing system. Context can be defined as "any information that can be characterized situation of an entity" [DA00]. The definition means that any piece of information that can be potentially used to pervasive computing system is a part of system context. For example, location, time and activity can be parts of context. Pervasive computing system is context aware "if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."[DA00]. Any pervasive computing system is context aware to some extent.

Situation can be defined as "external semantic interpretation of sensor data"[YD12]. In this definition *interpretation* means that from computational perspective situation is a formula that takes sensor readings as input and returns inference result as output. *Semantic* in the definition means that "situation assigns meaning to sensor data". *External* means "from the perspective of application, rather than from sensors", i.e. situation awareness functionality aims to benefit higher level applications. For example, application that automatically set profile of the phone might benefit from situations like *Noisy* or *InAMeeting*. Application that monitors and logs health of a user can benefit from situations *Hypertension*, *Tachycardia* and *UserFalls*. If the situation is of no use to any application, there is no reason to infer it at all.

From the perspective of two groups of definitions, related to context awareness [DA00] and to situation awareness [YD12], it can be concluded that situation awareness is the part of context awareness, which provides the uppermost layer of context generalization – generalization in terms of meaning of context.

The concepts related to situation awareness are activity recognition and location awareness. In pervasive computing human activity recognition aims to "recognize common human activities in real life settings"[KH10b]. The examples of recognized activities are locomotion like *UserSitting*, *UserStanding* or *UserWalking* [BI04], simple actions like "Opening door" or "Taking cup", or even complex actions like "Cooking" or "Cleaning"[KH10b]. The example activities can be viewed as "semantic interpretations of sensor data". Therefore, in pervasive computing activity recognition and situation awareness significantly overlap, and their common part is interpreting the sensor information in terms of its general meaning.

Location is one of the most important components of context. From pervasive computing perspective location awareness can be viewed as an aspect of context awareness, responsible for inferring and utilizing location information of users and objects in pervasive computing system. Location awareness overlaps with situation awareness on high levels of generalization. Generalizations of location like "At home", "In the office" or "At friend's place" belong to the field of location awareness. However, those generalizations also are "external semantic interpretation of sensor data"[YD12], i.e. situations.

Still situation awareness is not restricted to location awareness or activity recognition. For example, consider a wearable healthcare system that can detects situations like

Tachycardia or *Hypertension*. Generalization capabilities of the system are examples of situation awareness, but not an example of activity recognition or location awareness.

For more details about foundational aspects of context and context awareness an interested reader is referred to the paper by Dey and Abowd [DA00], which inroduced widely accepted definition of context and context awareness. The article by Ye et al. [YD12] provides a detailed introduction to the topic of situation awareness, as well as a comprehensive survey of situation awareness techniques. For the detailed information about activity recognition an interested reader is referred to the article by Kim et al. [KH10b].

Next section addresses situation definition methods in more details.

2 Defining Situations

From computational perspective every situation is described by a model, which takes raw or preprocessed sensor readings as input and produces reasoning result as output. Reasoning result might be probability of situation occurrence, fuzzy confidence or Boolean answer that identifies whether the situation is recognized or not. Obtaining the model of situation is one of the main challenges of situation awareness. The model should give correct reasoning results (i.e. it should not misinterpret sensor readings) and also it should be computationally feasible for runtime use (i.e. reasoning should not be too slow). We can obtain situation definitions in several ways.

1. Situations can be manually defined using the expert knowledge of the subject area.
2. Situations can be learned from labeled data.
3. Situations can be learned from unlabeled data.

From the perspective of entire pervasive system the approaches are not mutually exclusive. Situation aware system can include a multitude of situations, each of which is obtained by different method. Three approaches to situation have their own benefits and challenges. Next sections describe the methods to define the situations in more details.

2.1 Deriving Situations from Expert Knowledge

Sometimes human expert can just compose the formula by hand. For example, the formula of a situation, which takes blood pressure sensor readings as input and returns confidence in situation *Hypertension*, is relatively clear [DZ08]. Manual definition might be very efficient, but it is prone to human errors and restricted only to the formats that can be manually defined. For example, human expert can manually define a fuzzy set [DZ08], but manually defining the coefficients of neural network [Ha09] is often practically not possible.

This section overviews several approaches to situation awareness, which rely on manual definition of situations by human experts. This section introduces situation awareness concepts based on propositional and first order logic, belief function theory and Dempster-Shafer approach fuzzy logic and ontologies. This section also overviews spatial representation of context and situations.

2.1.1 Logic-based Approaches to Situation Awareness

Russel and Norvig [RN09] describe logic as “a general class of representation to support knowledge-based agents”. Logic allows the system not only to store the knowledge, but also to reason on it and properly infer new facts out of existing data.

One of the types of logic extensively used in situation awareness is propositional logic.

According to Kleene [Kl02] propositional logic is "a part of logic that deals only with connections between the propositions, which depend only on how some propositions are constructed out of other propositions that are employed intact, as building blocks, in the construction". The basic propositions, for which their internal structure can be ignored, are called prime formulas. In situation awareness scenario prime formulas can be, for example, *UserInTheLivingRoom*, *UserInTheKitchen*, *TVisON*. More complex propositions are called composite formulas, and they are defined as operations over other propositions. Composite formula in situation awareness can look like, for example, S*omeoneInTheLivingRoom*|*SomeoneInTheKitchen*|*SomeoneInTheHall* – there is either someone in the living room, or someone in the corridor, or someone in the kitchen. Each proposition is either true or false, but the value of particular proposition is not always known. In situation awareness scenarios the value of proposition can be known directly either from sensor values (e.g. pervasive system can detect that *TVisON*) or from expert knowledge and common sense (e.g. expression like (S*omeoneInTheLivingRoom*| |*SomeoneInTheKitchen*|*SomeoneInTheHall)*→ *SomeoneInTheHouse* can be asserted as true – if someone is in the kitchen, corridor or hall, it means that there is someone in the house). Propositions, for which the values are not asserted explicitly, can be inferred from the propositions with a known value.

While propositional logic can reason only about facts (i.e. prime formulas) and their relationships (i.e. composite formulas), the language of first order logic is built around facts, objects and relations. It is achieved by using quantifiers and predicates. Quantifiers include existence quantifier (∃) and universal quantifier (∀). An example of predicate in situation awareness scenario can be *Room(X)* (whether some object *X* is a room), *IsAt(X,Y)* (whether some object *X* is at a place *Y*), *User(X)* (whether some object *X* is a user of situation aware pervasive system).

The model in first order logic contains the following elements [RN09]:

1. Domain. Domain is the set of object that the model contains. For example, in pervasive comnputing domain can contain the objects corresponding to different users, rooms, appliances.

2. Relations between objects. Relations take one or more objects as arguments and produce Boolean output. For example, there can be a unary relation *User(X)* to determine whether object *X* is a user, unary relation *Room(X)* to determine whether the object *X* is a room or binary relation *IsAt(X,Y)* to determine whether user *X* is at room *Y*.

3. Knowledge base. Like in propositional logic, knowledge base is a set of sentences. Once the model is specified, the language of first order logic allows making certain assertions. First order logic allows using universal quantifier and existence quantifier in order to express the properties of collections of objects. For example, the sentence can be ∀ *X*, ∀*Y, User(X) & Room(Y) & IsAt(X,Y) & TvOn(Y)* → *WatchingTV(X)* – for any user *X* and room *Y*, if user *X* is at room *Y* and the TV in room *Y* is on, then user *X* is watching TV.

Once the model is specified, it can be used to infer new facts and answer the queries.

For more information about predicate logic (which incorporates first-order logic) refer to Kleene [Kl02], for more specific information about first order logic refer to Enderton [En01] or Russel and Norvig [RN09][RN06]. For temporal logic refer to the book [CG99]. Situation awareness systems based on fuzzy logic are discussed in the next section.

Logic-based solutions were used multiple times in context awareness and situation awareness systems. Henricksen and Indulska [HI06] proposed graphical context modeling language, and defined situations as logical expressions over the context model. An example of a situation "person is occupied" in provided in expression (1) (the expression is quoted from [HI06]).

```
Occupied(person) :
∃t1,t2,activity, such that
      engagedIn[person,activity,t1,t2]                                    (1)
      (t1 ≤ timenow() ∧ (timenow() ≤ t2 ∨ isnull(t2))∨
      (t1 ≤ timenow() ∨ isnull(t1)) ∧ timenow() ≤ t2)∧
      (activity = “in meeting” ∨ activity = “taking call”)
```

The meaning of situation definition (1) is following. The person is occupied, if there exist an “in meeting” of “taking call” activity for that person (line 6 of expression (1)). Lines 4 and 5 of expression (1) effectively mean that activity should start before current time and end after current time, but line 4 allows the ending time of an activity to be unspecified, while line 5 allows start time of an activity to be unspecified. However, either starting or ending time should be specified.

The permitted assertion included equality, inequality and relation (like “engagedIn[person,activity,t1,t2]” from expression 2). For evaluation purposes the use of quantifiers was restricted: the definition of every situation could begin with multiple existence quantifiers or with multiple universal quantifiers. Possible relations between different elements of context were defined in context model, which was designed by the means of context modeling language.

Seng W. Loke [Lo04b] proposed logic-based context awareness and situation awareness system for pervasive computing. The proposed system aimed to answer two types of queries:

1) Given an entity (which can be user, device or software agent) possible situations and contextual information (sensor readings and results of their processing), determine what situations are occurring.

2) Given a situation, an entity and contextual information, determine if the situation is occurring.

In order to design a system, which could handle those queries, the author proposed LogicCAP (Logic programming for Context Aware Pervasive application) – an extension of Prolog language with an operator that can handle type (2) quaries (see [RN09] for more details on Prolog). Type (1) queries could be handled by executing type (2) queries for all involved situations. The situations were defined in terms of rules. For example, the situation *WithSomeone(person, place)* could be defined according to expression (2) as a sufficient condtion and expression (3) as a necessary condition (example adapted from [Lo04b]).

```
If (Location(person, place))&&
  (PeopleInRoom(place,N))&&                                           (2)
 (N > 1)
 then WithSomeone(person, place)
```

```
If WithSomeone(person, place) then
  (Location(person, place)),
  (PeopleInRoom(place,N)),                                            (3)
 (N > 1)
```

Expression (2) asserts that if a person is at some place, there are N people at that place and that numer of people is more than 1, then it means that person is not alone in that room. Expression (3) asserts that if a person is not alone in a room, then there should be more than

1 person in that room. The argumets within condition can be sensor readings or other situations. Apart from inferring the situations, the same kind of rules was used to determine necessary actions that pervasive system should take.

Augusto et al. [AL08] proposed logic-based approach to context modeling in smart home systems. The authors represented expected behavior in terms of rules. However, they augmented logic with additional operators to represent temporal dependency. Tose operators included ANDlater (one condition is satisified later than the other) and ANDsim (both conditions are ssatisfied simultaneously). Expression (4) shows the rule to detect the situation "occupant fainted". The example is adapted from the paper [AL08].

IF at_kitchen_on ANDlater tdRK_on ANDlater no_movement_detected (4)
THEN occupant_fainted

Rules for actions can be composed in a similar manner.

To summarize, many situation aware systems in pervasive computing use logic-based knowledge representation and reasoning. A distinct type of logic is fuzzy logic, which is overviewed in the next section.

2.1.2 Fuzzy Logic for Situation Awareness

Fuzzy logic originates from the works of L. Zadeh [Za65], who introduced the concept fuzzy sets. Fuzzy set is "a class of objects with a continuum of grades of membership"[Za65]. In an original set an object either belongs to the set or does not belong to it. In a fuzzy set every object belongs to a set with a certain degree of membership, which can vary from 0 to 1. Zadeh concludes that in real life objects sometimes do not have precise criteria of membership. The author argues that the classes like "tall men" do not consititue sets in original sense, but still play an important role in human thinking. Fuzzy sets provide the tool to describe the classes like "tall man", "cold day", "dark place", and fuzzy logic allows additional reasoning with these classes.

Every fact in fuzzy logic is treated as a degree of belonging to some fuzzy set. Figure 1 describes an example membership function for a certain room to be a member of the set "dark room". As figure 1 shows, membership function depends on the iluminance in the room. If illuminance is less than 350 Lx, the room is definitely dark (membership function 1). If the illuminance is above 500 Lx, the room is definitely not dark (membership function 0). If the illuminance is between 350Lx and 500Lx the room is considered to be somewhat dark, and degree of membership varies.

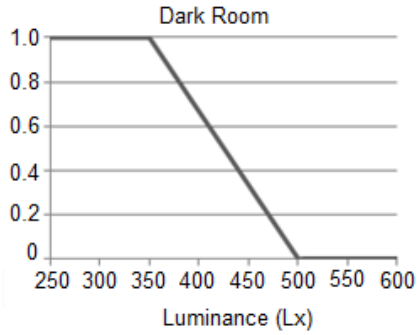


Fig. 1. An example of a membership function

The shapes of membership function might vary. For example, the shape of membership function "temperature is approximately 25°C" will most likely be triangular depending on temperature with a peak at 25°C. Or, for example, the shape of the membership function "morning" will most likely have trapezoidal shape depending on time of the day. Popular shapes of membership functions are presented in [HM93]. From situation awareness perspective genralizations in terms of fuzzy set membership can already be viewed as a situation.

Fuzzy logic can work with relations between different classes. For example, smart home developer can assert that the room is suitable for work if it is silent and not dark. Therefore, it can be asserted that *RoomSuitableForWork⇔SilentRoom&(¬DarkRoom)*. Those relations can be described in terms of Zadeh operators [Za65] (expression (5)). Zadeh operators represent interactions between fuzzy sets, e.g. degree of membership in an intersection of fuzzy sets *SilentRoom* and *WellIlluminatedRoom* is the minimum of two degrees of memebrship.

$$\begin{aligned} &AND: A \,\&\, B \;=\; min(A,B) \\ &OR: A \mid B \;=\; max(A,B) \\ &NOT: \neg A \;=\; 1 - A \end{aligned} \tag{5}$$

For more details on fuzzy logic, fuzzy control and decision making refer to the works [Pi01][RN09][Za65][HM93].

Some situation awareness systems in pervasive computing are based on fuzzy logic. An example of fuzzy logic-based approach to situation awareness is the work by Anagnostopoulos et. al [AN06]. The authors introduced a framework for context awareness and proposed situation reasoning mechanisms. A situation in [AN06] is viewed as a conjunction of context features (see expression (6), quoted from [AN06]).

$$\bigwedge_{i=1}^{N} context(x_i, user) \rightarrow IsInvolvedIn(Situation, user), \;\; N > 1 \tag{6}$$

In a similar manner the framework [AN06] decides, what action the system should take if certain situations take place.

The paper [MS02] used fuzzy logic to provide context aware control to mobile terminals. On the first stage of context processing the proposed system extracted context features out of raw measurements. On the subsequent step the features underwent fuzzy quantization (i.e. representation in terms of fuzzy set membership). One of the aspects of fuzzy quantization was recognition of user's activities: whether the user walks and whether the user runs. Another aspect was generalization of sound level (silence, modest sound, loud sound) and environment illumination level (bright, moderate or dark). As a result, the system was able to control the settings of mobile terminal based on the perceived conditions.

Cao et al. [CX05] used fuzzy logic for pervasive service adaptation. The authors applied fuzzy quantization to the characteristics like network delay, clock rate or free space in the RAM. The proposed system used fuzzy logic in order to adapt the services like chat service or e-mail service to current condition.

Acampora et al. [AG10] used fuzzy logic to actuate proper services in ambient intelligence systems. Sensor information was generalized using fuzzy quantization. It produced such generalizations like "Internal temeprature is high" or "time is evening",

which can be viewed as situations. Subsequent logical inference was used to infer the suitable adaptation policy. It should be noted that both the papers [CX05] and [AG10] used the term "context situation", which they defined as "a combination of context information"[CX05]. This term differs from the term "situation" in this thesis and should not be confused with it.

Fuzzy logic is popular and powerful method both for generalization of context information and for adaptation in pervasive computing system. Next section discusses situation awareness using ontologies – a popular knowledge engineering concept, which is closely related to logic-based inference.

2.1.3 Ontologies for Situation Awareness

Ontologies in computer science provide generic domain independent way to represent, share and reason about knowledge [Gr93]. One of the most popular definitions of ontologies is provided by Gruber [Gr93]. Gruber defines ontology as "explicit specifications of a conceptualization". The author further explains the concept of ontologies by adding the notion of universe of discourse and the notion of vocabulary. The article [Gr93] defines universe of discourse as "the set of objects that can be represented" and vocabulary as "set of objects, and the describable relationships among them". Most important relations include "is-a" relation (e.g. accelerometer is a sensor) and "instance-of" relation (room A3304b is an instance of a room). As a result, the author proposes the following expanded definition of ontology: ontology is "a specification of a representational vocabulary for a shared domain of discourse — definitions of classes, relations, functions, and other objects"[Gr93]. An example of ontology is presented in figure 2. The ontology in figure 2 is a simplified situational context ontology, which was used in the article [AN06].

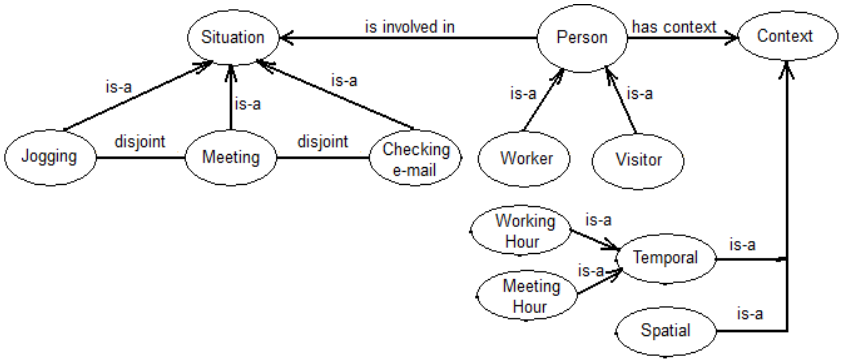


Fig. 2. An example of a situation awareness ontology

The ontology in figure 2 describes the realtions between different types of situations, aspects of context and information about users of the system. For example, when a meeting is added to the system, corresponding node is added below a "Meeting" node with "instance-of" relation. Ontologies allow inferring the facts. For example, it can be straightforwardly inferred that meeting hour is a part of context.

Ontologies provide techniques to share knowledge between several reasoning agents,

infer new information and detect incomsistencies in a knowledge base. For more details on ontological engineering, languages to describe ontologies and inference methods for ontologies, refer to the handlbook [St09].

Ontologies are widely used in context awareness and situation awareness in pervasive computing. Anagnostopoulos et al. [AN06] proposed ontology-based situation awareness technique, which was partically described in section 2.1.2 and earlier this section. Wang et al. [WZ04] introduced context ontology for logic based context reasoning ans situation awareness. Chen et al. [CF05] proposed SOUPA – Standard Ontology for Ubiquitous and Pervasive Applications. Dimakis et al. [DS07] proposed ontology-based mechanisms to provide necessary information to pervasive services. Ejigu et al. [ES07] proposed general purpose ontology-based context modeling system. For more details and more examples of ontologies in pervasive computing, refer to the survey by Ye et al. [YC07b] and the survey by Bettini et al. [BB10].

Next section introduces situation awareness methods based on combining the evidences. In particular, next section overviews the methods based on Dempster-Shafer approach and belief function theory.

2.1.4 Theory of Evidence for Situation Awareness

Theory of evidence [Sh76], also known as Dempster-Shafer theory [Sh76] or theory of belief functions [YL08], aims to combine the evidences and fuse the data from various senors [Sh76][RN09]. Also Dempster-Shafer approach aims to draw a distinction between uncertainty and ignorance. It is achieved by transitioning from probability that a proposition is true to the probability that the evidence supports the proposition. That measure is referred to as *belief function* [RN09].

The concept of belief function can be illustrated with a following example. Assume that there is an area, out of which 50% is known to be land, 30% is known to be water and the remaining 20% area is unknown. In that case, if a random point is uniformly picked in the area, the probability that this point is on the land is at least 0.5 (assuming that all unknown area is covered by water) and at most 0.7 (assuming that all unknown area is also land). Therefore, the belief in the fact that point is on the land is 0.5, and the plausibility that the point is on the land is 0.8. Belief and plausibility in the fact that the point is on the land can be written as [0.5; 0.8].

In order to formally define belief and plausibility and overview some advanced cases, consider a more complicated example. Now there are three mutually exclusive options: some area is either plain, or covered by water, or covered by hills. From the whole area 20% is known to be water, 25% to be plain, 15% to be hills. For 10% of the whole area it is known for sure that there is no water there, but it is unclear to what extent the terrain is flat and to what extent it is covered with hills. For the remaining 30% of the area nothing is known – it can contain water, plains or hills in any proportion. Once again, consider that a random point is unformly picked on that area.

The set {*Plain, Water, Hills*} from the example is referred to as frame of discernment. In pervasive conmputing the elements of frame of discernment are events or situations. The elements of frame of discernment should be mutually exclusive.

The set of all subsets is referred to as power set. For the example the power set is {*Water, Plain, Hills*} the power set will be {Ø, {*Plain*}, {*Water*}, {*Hills*}, {*Plain, Water*}, {*Plain, Hills*}, {*Water, Hills*}, {*Plain, Water, Hills*}}. Empty set ususally corresponds to

contradictive evidences. Size of a power set is 2^D, where D is the size of frame of discernment. Therefore, power set grows exponentiallywith the set of frame of discerment.

Mass function assigns value from the interval [0,1] to every element of a power set. In the example the mass *m({Plain})* is 0.25, the mass *m({Water})* is 0.2, the mass *m({Hills})* is 0.15, the mass *m({Plain, Hills})* is 0.1 and the mass *m({Plain, Water, Hills})* is 0.3. The remaining masses are 0.

The formal definition of belief and plausibility is defined in expressions (7) and (8) respectively.

$$bel(A) = \sum_{B \subseteq A, B \neq \emptyset} m(B) \quad (7)$$

$$pl(A) = \sum_{(B \cap A) \neq \emptyset} m(B) \quad (8)$$

As follows from expression (7), belief for some element A of the power set is a sum of masses of all the other elements of a power set, which are contained by A (excluding the empty set). For example, consider the belief that a randomly picked point is on the plain. The sum will contain only one summand – *m({Plain})*. Therefore, *bel({Plain})* = 0.25. As for the belief that a randomly picked point is either on the plain or in the hills, according to expression (7) that belief consists of *m({Plain})*, *m({Hills})* and *m({Plain, Hills})*. Therefore, *bel({Plain, Hills})*=0.25+0.15+0.1=0.5. Belief in the fact that randomly chosen point is either on the land, or in the hills, or on the water is equal to 1. One of those options is definitely true, and formula (7) reflects that. The belief *bel({Plain, Hills, Water})* is the sum of all the masses except for the mass of empty set (that is 0 is our case).

Formula (8) shows that plausibility of some element A of a power set is equal to the sum of masses for all the sets, which have non-empty intersections with A. For example, the plausibility that some random point is on the plain will consist of *m({Plain})*, *m({Plain, Hills})*, *m({Plain,Water})* and *m({Plain,Water,Hills})*. Therefore, the plausibility *pl({Plain})* = 0.25+0.1+0.3=0.65. The plausibility that a randomly picked point is in the hills or on the plain consists of evey mass except *m({Water})* and the mass of empty set. Therefore, the plausibility *pl({Plain,Hills})=0.8*.

Multiple evidences can be combined in a following manner. Several evidences correspond to several mass functions over the same power set. A single mass function, corresponding to the combined evidences, is calculated using Dempster's rule of combination. Dempster's rule of combination is presented in expression (9).

$$m(C) = \frac{\sum_{B \cap A = C \neq \emptyset} m_1(A) * m_2(B)}{1 - \sum_{B \cap A = \emptyset} m_1(A) * m_2(B)} \quad (9)$$

In expression (9) the terms m_1 and m_2 refer to the mass functions that come from two evidences that need to be combined. The function *m* refers to the resulting mass function. If the conflicts between evidences should be ignored, then expression (9) is applied to all the elements of a power set except for Ø, and the mass *m*(Ø) is set to 0. The term $\sum_{B \cap A = \emptyset} m_1(A) * m_2(B)$ in the denominator in expression (9) is the measure of conflict between two pieces of evidence.

For more details on Dempster-Shafer approach, its extensions and modifications refer to [Sh76][RN09][Ra07][YL08].

Pervasive computing systems usually contain a variety of sensors, which provide multiple evidences for situations. Some pervasive computing systems used Dempster-Shafer approach to combine the evidences and generalize sensor information.

Padovitz [PZ06][Pa06] proposed to use Dempster-Shafer theory in combination with context spaces approach in order to fuse context information and achieve efficient situation awarenes. Hong et al. [HN09] proposed to use Dempster-Shafer approach to recognize activities of daily living in a smart home. Zhang et al. [ZG10] proposed CRET approach (Context Reasoning using extended Evidence Theory). In order to account for possible conflicting evidences, the authors proposed new evidence selection and conflict resolution strategies. McKeever et al. [MY10] proposed a method to recognize the activities of inhabitants in smart home. As part of the approach, the authors introduced an extension of Dempster-Shafer theory to take into account temporal aspects of activities. Dominici, Pietropaoli and Weis [DP11] used theory of evidence for sensor fusion in a smart home environment. Evidence theory allowed combining data from multiple sensors and inferring contextual abstractions such as presence of someone in the room. Situation awareness was achieved by combination of evidence theory and context spaces approach [PL08b].

In pervasive computing Dempster-Shafer approach has been successfully used because it allows fusing the information from multiple sensor readings and context features. Next section introduces a way to fuse sensor readings and context features by representing possible context as multidimensional space. This method can be combined with other situation awareness approaches, and it has been successfully used in conjunction with with fuzzy logic [DZ08] and Dempster-Shafer approach [PZ06] [Pa06].

2.1.5 Spatial Representation of Context and Situations

Spatial representation of context and situations emerges from a straightforward idea that a set of important context parameters can be represented as a vector of values. It can be illustrated by a following example. Consider a context of a room in a smart home. That context includes number of people, level of noise and the position of the door (open or closed). Other context characteristics are omitted for the purpose of simplicity. The considered parameters can be either measured directly (e.g. the level of noise), or inferred out of sensed information (e.g. number of people can be estimated by processing indoor positioning information for all people in the building). At any particular time context features can be combined into a single vector. In the example a vector can look like [3; 40 dB; Closed] – there are three persons in the room, noise level is 40 dB and the door is closed. Note that elements in a vector can as well be non-numeric.

The set of all possible vectors can be viewed as multidimensional space. A multidimensional for the considered example is proposed in figure 3. The point in figure 3 represents the ealier mentioned vector [3; 40 dB; Closed]. In some practical cases the position of the point can be unclear due to sensor uncertainty, or the value for some dimensions can be unknown due to sensor unavailability.

One of the most prominent methods for spatial representation of context is context spaces approach [PL04][PL08b][Pa06]. The context spaces approach extensively uses spatial metaphors to represent context and situations. A multidimensional space like the one presented in figure 3 is referred to as *application space* or *context space*. Every axis of application space is referred to as *context attribute*. So in the example context attributes are the number of people, noise level and door position. A point in a multidimensional space is referred to as *context state*. Therefore, at any moment context of pervasive computing system corresponds to some context state, and functioning of pervasive computing system over time can be characterized by a trajectory of context state in an application space.

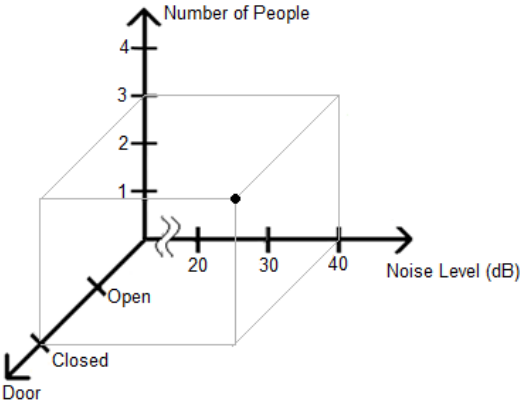


Fig. 3. Context Spaces Approach – an example

In order to reason about situations, context spaces approach introduced the concept of *situation space*. The confidence in a situation was calculated as weighted sum of contributions of different context attributes. Generic formula of a situation is presented in expression (10).

$$conf_S(X) = \sum_{i=1}^{N} w_i * contr_{Si}(x_i) \tag{10}$$

In expression (10) the term $conf_S(X)$ corresponds to the confidence that situation S is occurring (depending on the context state X), w_i corresponds to the weight of i-th context attribute and $contr_{S,i}(X)$ corresponds to the contribution of i-th context attribute to the situation S. For example, for the situation *Hypertension* [DZ08] the contributing context attributes can be *SistolicPressure* and *DiastolicPressure*, both with the weights 0.5. Weights sum up to 1, and the value of contribution function is between 0 and 1. Therefore, confidence is between 0 and 1.

Generic contribution function for numeric context attribute is presented in expression (11). For numeric context attribute different contribution values are assigned to different intervals of a context attribute. For non-numeric context attributes different contribution values are assigned to different non-numeric values of a context attribute.

$$contr(x) = \begin{cases} a_1, x \in (b_1, b_2] \\ a_2, x \in (b_2, b_3] \\ \ldots \\ a_m, x \in (b_m, b_{m+1}] \end{cases} \tag{11}$$

Situation algebra allows reasoning about the relationships between situations. For example, situation algebra allows finding the confidence in the fact that user is either walking or running (*UserWalking*|*UserRunning*). The operations AND, OR and NOT of situation algebra are based on Zadeh operators [Za65] (expression (5)), and more complicated logical expression can be claculated using operators (5) as basis.

The provided definition of situation is flexible enough to represent a broad class of real life situations. Moreover, it allows fast reasoning algorithms and still the concept of

situation space is clear enough for the situations to be composed by human expert.

Delir et al. [DZ08] proposed fuzzy situation inference - an extension of the context spaces approach. The main difference was that instead of original contribution function (11) the authors used a degree of belonging to a fuzzy set.

Padovitz et al. [PZ06], the developers of original context spaces approach, proposed an extension to context spaces approach, which combined spatial representation of context with situation reasoning based on Dempster-Shafer theory.

A different way to combine context spaces approach and evidence theory was proposed by Dominici et al. [DP11]. The authors used the methods of evidence theory to estimate context features, and then applied extended context spaces-based situation reasoning.

Another example of spatial representation of context is the work by Anagnostopulous et al. [AM05]. The authors represented possible context as a multidimensional space and current condition as a point in multidimensional space. However, in contrast with context spaces approach and subsequent works, the work [AM05] did not generalize context in situations. The paper [AM05] proposed context prediction by extrapolating the trajectory in the space of context. The similar principle in application to context spaces approach was proposed by Padovtiz et al. [PL07].

Although the term "context space" is specific to context spaces approach, graphical representation of context provides useful inutitions in many cases. Throughout the thesis the space formed by context features will be referred to as context space, even outside the context spaces appoach.

This concludes the topic of defining situations using expert knowledge. Defining situations manually can be very efficient, but it is not always possible. Learning the definition of a situation is often a viable option if a developer cannot define the mapping between sensor readings and situation.

2.2 Learning Situations from Labeled Data

Sometimes the list of situations is known, and the sensors provide enough information for reasoning, but still the formula is unclear. For example, consider a wearable computing system with multiple accelerometers and orientation sensors attached to the smart clothes (example close to [BI04]). The provided information might be enough to detect situations like *UserStanding*, *UserSitting* or *UserWalking*, but the formulas of those situations are not clear. The solution is to design an experiment and collect labeled data – sensor readings annotated with information about situation. In the example the solution is to watch testers wearing the system in a lab, and log both the sensor readings and occurring situations. Real occurring situations can be extracted manually, for example, from camera image of the tester. Once enough labeled data are collected, developers can use multiple supervised learning methods [RN06] to extract the formulas of situations.

This section contains some practical examples of that approach. Following subsections provide brief overview of supervised learning techniques, which were used to infer situations in pervasive computing. In particular, several subsequent sections discuss learning the situations in the form of Bayesian networks [RN09] – one of the most popular probabilistic graphical formalism.

2.2.1 Naïve Bayesian Approach for Situation Awareness

Naïve Bayesian approach is a relatively simple, yet very efficient way to combine the evidence from multiple sources. This method relies on a strong assumption, that evidences are conditionally independent from one another having the situations [RN09]. Consider the

following illustration: a smart home system evaluates the presence of people in the room. The used sensors include sound sensor and pressure sensors on the floor. In order to mitigate possible challenging scenarios (like sound from TV left on, which can lead to evidences of presence from sound sensors), pervasive system should combine the evidences from multiple sensors. So, the involved situation is *presence*, and the task is to combine the evidences and calculate the probability of presence of one or more persons in the room. First step to solve the task is applying the rule of Bayes. Rule of Bayes, applied to illustration scenario, is presented in formula (12).

$$P(presence \mid sound, pressure) = \frac{P(sound, pressure \mid presence) * P(presence)}{P(sound, pressure)} \quad (12)$$

In the illustration scenario it is safe to assume that when there is no one in the room, the readings of pressure sensor and sound sensor are independent from each other. The same applies for the case when someone is in the room. The resulting conditional independence is presented in formula (13).

$$P(sound, pressure|presence) = \\ = P(sound|presence) * P(pressure|presence) \quad (13)$$

Formula (13) illustrates the main assumption of naïve Bayesian approach – evidences are independent give the situation. Learning the probabilities $P(sound|presence)$, $P(pressure|\neg presence)$, $P(sound|\neg presence)$ and $P(pressure|presence)$ requires much less training data than learning the joint probability $P(sound, pressure|\ presence)$ and $P(sound, pressure|\ \neg presence)$. The remaining terms of formula (12) can be obtained in a following manner: $P(presence)$ can be inferred directly from the training data (as well $P(\neg presence)$). As for denominator of (12), it is a sum of numerator (12) and similar numerator for probability of non-presence.

The benefits of naïve Bayesian approach include efficient learning, as well as fast runtime inference. Every learning task for naïve Bayesian approach requires learning a distribution function of one variable. The distribution can be learned using maximum likelihood approach. Exact learning method depends on chosen distribution function [RN09]. More variables (sensors or context parameters) the task has, more training data is required to learn the distribution and, hence, more benefit naïve Bayesian approach provides. However, failing to satisfy a very strong assumption of conditional independence can lead to problems.

Once the probabilities are learned, formula (12) can be used at runtime to calculate the probability of a situation given the sensor data.

Naïve Bayesian approach was used in multiple pervasive computing tasks. In the paper [BI04] authors used naïve Bayesian approach to infer user locomotion from acceleration data. The authors used five wearable accelerometers to recognize the activities of the user. Twenty recognized activities included walking, riding the escalator, vacuuming, lying down. The authors used a following assumption: having the situation, the sensor readings are distributed normally and independently of each other. The authors learned probability distributions out of training data and tested the performance in the lab. However, the performance of naïve Bayesian was unsatisfactory. According to authors' analysis, conditional assumption was not satisfied in practice.

In the paper [MB04] authors used Naïve Bayesian approach to detect activity and availability of the user. Possible activity included using PC, using PDA, talking on the phone, meeting, walking. The authors used naïve Bayesian method to extract this information from the observable values like PC usage, ambient sound, iPAQ location, time of the day. Possible avalability values were "available for a quick question", "available for

a disscussion", "to be available soon" and "not available". The paper [MB04] proposed to infer availability value using naïve Bayesian approach. The considered observable information included user's activity, user's location and time of the day.

In the paper [TI04] authors aimed to explore whether activities of daily living can be inferred using massive amounts of simple sensors. One of the considered options was naïve Bayesian approach. The activities included "preparing breakfast", "watching TV", "listening to music".

An example of naïve Bayesian approach applied to pervasive computing is presented in the paper [KK07]. The authors applied naïve Bayesian approach to recognize activities of daily living in the residence for elders. As a baseline model they used Naïve Bayesian approach, and then augmented it with time dependencies and transformed into dynamic Bayesian network. In their subsequent work [KK08] the authors proposed CARE (Context Awareness in Residence for Elderly) system. The activity recognition techniques once again included naïve Bayesian approach.

Naïve Bayesian model can be viewed as a very simple case of Bayesian network. More general examples of Bayesian networks are described in the next sections, as well as the examples of Bayesian networks used in situation awareness.

2.2.2 Bayesian Networks for Situation Awareness

The Bayesian network is a direct acyclic graph where every node is associated with a fact and every directed edge represents the influence on one fact by another. Nodes of the Bayesian network have local Markov property: each variable is independent of its non-descendants given its parents. For basic information on Bayesian networks refer, for example, to Russel and Norvig [RN06][RN09].

A classic example of a Bayesian network is the sprinkler and rain network depicted in the figure 4.

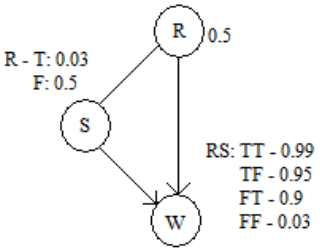


Fig. 4. Bayesian network example

The following interpretation begins by examining three facts – whether there was rain (node R), whether the sprinkler was working (node S) and whether the grass is wet (node W). Every node has probability distribution depending on its parents; for example, if it is both raining and the sprinkler is working, then the grass is wet with a probability of 99 per cent; if it is just raining – a 95 percent probability; if just the sprinkler works – 90 per cent probability; and if there is no rain and sprinkler is off – a probability of three per cent. Nodes with no parents have only prior probability (for rain it is 50 per cent). Some facts are directly observed (e.g., we see or hear that the sprinkler is working). The system can compute the posterior probabilities of unobserved facts using either the formula of Bayes (up the graph – whether it is raining) or direct probability calculations (down the graph – whether the grass is wet).

A learning task is quite common for Bayesian networks. Users sometimes do not have complete information about the network and need to infer probability distributions of the nodes (parameters learning task) or even the structure of Bayesian network graph itself (structure learning task). For more details refer to the book by Russel and Norvig [RN06][RN09].

Ye et al. [YC07a] proposed a concept of situation lattice. In brief situation lattice can be defined as follows: "Situation lattice L, is defined as L = (S, ≤), where S is a set of situations and the partial order ≤ is a generalisation relation between situations"[YC07a]. The authors mentioned that the concept of situation lattice was inspired by the concept of lattice in linguistics. The authors also noted that the semantics of situation lattice is effectively captured by Bayesian network, and that situation lattice can be converted to Bayesian network for further reasoning by a straightforward algorithm.

Among multiple other methods in the paper [DP07] the authors used Bayesian networks for context reasoning in smart homes. As an example authors used automatically inferred Bayesian network, which connected the characteristics like motion detection, time of the day, blinds position in a room, luminosity. That Bayesian network also included situations like user presence in the room or user actions.

In the paper [ZS11] authors used Bayesian networks to detect when the user falls. Falling was detected using sensors like gyroscopes and accelerometers, and additional Bayesian network was used to validate the results. In the Bayesian network the node "Fall detection alarm" (i.e. that sensors detected the fall) was a child node to the "Fall" node (i.e. that the fall has really occurred). The parents of "Fall" node were the nodes corresponding to physiological condition, physical activity and location. Physiological condition in turn depended on health record and age of a user. The observable information was user profile and the fact whether sensors detected the fall or not. Using observable information Bayesian network could determine whether the fall has really occurred.

For more examples of Bayesian networks in situation awareness refer to survey by Ye et al. [YD12]. A distinct type of Bayesian network is dynamic Bayesian network (DBN). Dynamic Bayesian networks allow capturing time dependencies between the variables, and multiple works used DBN for situation awareness [KK07][DP09][IN09][LO11] and situation prediction [AZ98][PP05] task.

2.2.3 Dynamic Bayesian Networks for Situation Awareness

A dynamic Bayesian network (DBN) is an extension over of Bayesian network that takes timing into account. Consider time being discrete (t=1,2,3….) the dynamic Bayesian network can be defined as a pair of B_1,$B_{->}$, where B_1 is the Bayesian network which defines all prior probabilities on time t=1 and $B_{->}$ defines several-slice (as usual, two-slice) temporal Bayesian network, where all time dependencies are represented in terms of directed acyclic graph. For example, it can look like this (see figure 5).

Dashed lines in Figure 5 represent temporal dependencies. B_1 is the initial graph of t=1 with all the necessary starting probability distributions, $B_{->}$ being a combined graph of t=1 and t=2 (with all the necessary distribution functions).

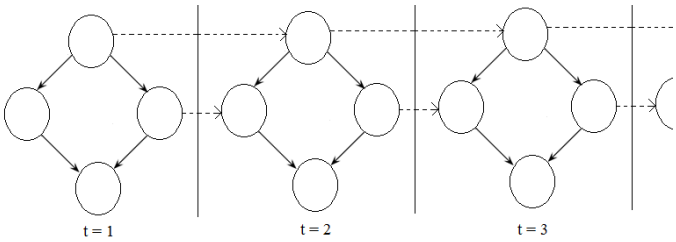


Fig. 5. Dynamic Bayesian network example.

Learning tasks in dynamic Bayesian networks (DBNs) are the straightforward generalisation of learning tasks for simple Bayesian networks: parameter learning and structure learning. In context prediction the tasks structure of a dynamic Bayesian network is usually known and the system needs to infer parameters. For more information on dynamic Bayesian networks see, for example, the work [RN06][RN09].

Dynamic Bayesian networks have been used in practice for situation awareness and activity recognition purposes. For example, in the paper [KK07] the authors used dynamic Bayesian network to recognize activities in the residence for elderly. The authors started with naïve Bayesian model for activity recognition and then procceded to the connection between activities – activity on the previous step influenced activity on the subsequent step, and the dependency formed dynamic Bayesian network.

The paper [IN09] proposed using dynamic Bayesian network to recognize activity from sensed interactions with objects. The system was designed to facilitate the work of a nurse. The authors used RFID sensors to detect interaction of nurse with tool and materials: RFID tags were attached to the objects, while RFID reader was munted on a wrist of a user. The authors used dynamic Bayesian network to fuse sensed information and infer the ongoing activity. For different activities of drip injection task the paper [IN09] claims over 95% recognition accuracy.

Dimitrov et al. [DP07] used dynamic Bayesian networks in their context reasoning framework. As the authors noted, Bayesian network, which they also used, was unable to reason over time, and they resorted to dynamic Bayesian network to encode timing dependencies. Inferred situations included, for example, “user leaving the house”[DP07].

In the paper [LO11] authors used dynamic Bayesian network for human activity recognition. The authors used the dataset collected by van Kasteren [KA08]. During dataset collection a set of state-change sensors was attached to multiple places in a smart home including doors, cupboards and refrigerator. The recognized activities were "Leave house", "Toileting", "Showering", "Preparing breakfast", "Preparing dinner", "Preparing a beverage", "Sleeping" and “Idle”. In the paper [LO11] the authors achieved ~80% precision and recall on the mentioned dataset. However, dynamic Bayesian network was used only as a benchmark for comparison, and the main focus was on the techniques featuring sliding window and decision trees, which gained over 90% of precision and recall.

Dynamic Bayesian networks conclude the overview of Bayesian network-based approaches to situation awareness. The next sections discuss logistic regression and support vector machines. Those two approaches usually separate context space (see section 2.1.5) by a hyperplane into two parts – the part where the situation occurs and part where situation does not occur.

2.2.4 Logistic Regression for Situation Awareness

As previous sections have shown, the task of situation inference can be viewed as the task of estimating the probability of a situation. Logistic regression emerges from the task of estimating the probability, and enables a practical solution both for learning situation definitions and for inferring situations during the runtime. The basic concepts of logistic regression are presented, for example, in the book [RN09][HL00].

Usually logistic regression relies on the use of sigmoid function. The formula of sigmoid function is presented in the expression (14).

$$sigmoid(x) = \frac{1}{1+\exp(-x)} \tag{14}$$

The plot of sigmoid function is presented on figure 6. The function can be skewed or/and moved by using linear function a**x*+b as an argument instead of *x*. Sigmoid lies between 0 and 1 and, therefore, can represent probability.

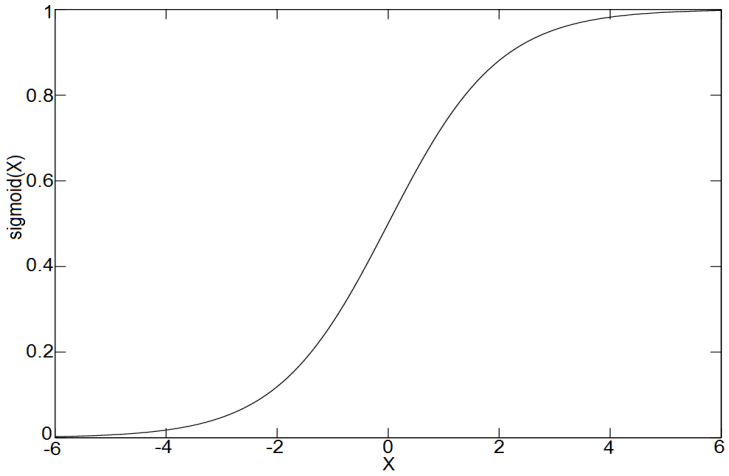


Fig. 6. Sigmoid function example.

Probabilistic estimation can be converted into Boolean decision. Usually if probability reaches some threshold (often 0.5) the situation is claimed to occur, while if the probability is below the threshold the situation is counted as non-occurring. Sigmoid is ascending function, so a threshold for sigmoid value effectively means a threshold for sigmoid argument. For example, sigmoid is greater than 0.5 when the argument is greater than 0.

Logistic regression aims to learn the probability model in the form of sigmoid function. Consider a multidimensional context space like the one presented in the section 2.1.5. The probability is estimated as a sigmoid of linear combination of context features: *sigmoid*(W^T*X+b), where X = [1 x_1 x_2 … x_N] is the vector of context state and W = [w_o w_1 … w_N] is the vector of weights. Usually an input vector starts with the value x_0 = 1 in order to gracefully incorporate the bias term. Just as described in previous paragraph, if we use the threshold 0.5, the situation is claimed to occur if W^T*X ≥ 0 and is claimed not to occur

if $W^T*X < 0$. However, W^T*X defines a hyperplane in multidimensional space, and expressions $W^T*X \geq 0$ and $W^T*X < 0$ define two sides of that hyperplane. To summarize, logistic regression approach separates multidimensional context space by hyperplane; if current context reading is on the side of the hyperplane where $W^T*X \geq 0$ the situation is counted as occurring, otherwise – as non-occurring. However, the challenge is to find a vector of weights, which fits the training data best.

A generic illustration is provided in figure 7. A line (in general case – a hyperplane) separates labeled data. Context readings, where the situation has occurred, are marked by **o**. Context readings, where the situation did not occur, are marked by **x**. Those are training data, which were used to construct the hyperplane. When new context state arrives at the runtime, it is tested against the hyperplane. If context state is on top-right of separating line (or exactly on the line), the situation is counted as occurring, otherwise – as non-occuring.

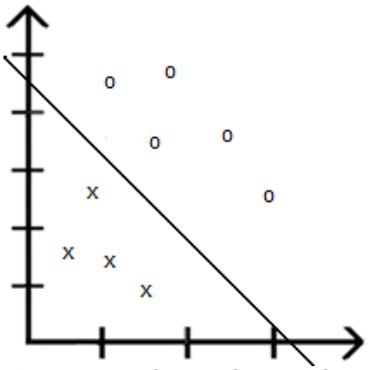


Fig. 7. Separating line in 2-dimensional context space

Coefficients of the separating hyperplane can be learned in the following manner. Assume that the pervasive system developers have M points of training data. For example, the developers could aim to infer situation "sitting", and collected some wearable accelerometer readings when the situation "sitting" has occurred and some wearable accelerometer readings when the situation "sitting" did not occur. Particular context readings used for training are denoted as X_i (where i is an index from 1 to M) and the corresponding label is denoted as Y_i. If the situation occurred (points **o** in figure 7) the real probability of occurence is one (Y_i=1). If the situation did not occur (points **x** in figure 7) the real probability of occurence is zero (Y_i=0). A frequently used error estimation for a single example is following (expression (15)).

$$cost(X_i, Y_i, W) = \begin{bmatrix} -\log\left(sigmoid(W^T * X^i)\right), if\ Y_i = 1 \\ -\log\left(1 - sigmoid(W^T * X^i)\right), if\ Y_i = 0 \end{bmatrix} \quad (15)$$

The total error estimation is the sum of estimations for evey learning example (formula (16)). The use of cost function (16) makes error estimation a convex function of weights, which makes optimization significantly easier.

$$J(W) = \sum_{i=1}^{M} cost(X_i, Y_i, W)) \quad (16)$$

The error function depends on the weights of sigmoid argument. This error function can be minimized using, for example, gradient descent (note that linear separability is not required for the method to work). The optimal weights, corresponding to mimum error W_{opt} = argmin(J(W)) are learned during the development of situation aware system. At the runtime those weights are incorporated into the system as constants. For every new sensor reading X probability of a situation occurrence can be estimated merely as *sigmoid*(W^T*X), and then this probability can be compared against the probability threshold.

For more information on logistic regression refer to the books [HL00] and [RN09].

Logistic regression was used for situation awareness in pervasive computing on multiple occasions. Kwapisz et al. [KW10] used cell phone accelerometers to infer the activities of a user. The system extracted a set of features out of raw accelerometer data. Those features were input of activity recognition algorithm, which needed to distinguish between walking, jogging, going up and down the stairs, standing and sitting. Logistic regression was one of the investigated activity recognition options, along with decision tree and multilayer perceptron. The performance of logistic regression varied a lot between different activities: it has achieved accuracy of 98% for jogging, but for going downstairs only 12% accuracy was achieved.

Al-Bin-Ali and Davies [AD04] proposed to use logistic regression for activity recognition purpose. The authors used three light intensity sensors – kitchen sensor, bathroom sensor and bedroom sensors. As a result, they were able to infer the activities like bathing, cooking, watching movie or sleeping. The accuracy was 60.8% when using bathroom sensor only, 98% whn using bathroom and kitchen sensors and 99.2% when using all thress sensors.

Ryoo and Aggrawal [RA09] used logistic regression to estimate the probabilities of high-level user activities. The system used computer vision as a sensor input, and then on lower level it hierarchically ecognized body parts positions, then posture, and then basic human gestures. On higher level logistic regression was used to to recognize activities and interactions between users (like “greeting” or “fighting”). Probability of activity was estimated as a value of sigmoid function, which used weighted sum of gesture features as an input.

Next section discusses support vector machines and their application to situation awareness task. SVM has some similarity with logistic regression, but it uses different criteria for constructing the separating hyperplane.

2.2.5 Support Vector Machines for Situation Awareness

Support vector machine (SVM) is a supervised learning method for linear and, whith some extensions, non-linear classification. Russel and Norvig [RN09] claim that SVM is currently the most popular approach for off-the-shelf supervised learning. SVM made its way into the context awareness and situation awareness fields as well.

Like logistic regression, SVM method learns from labeled data and builds classifier, which distinguishes situation occurrence from situation non-occurrence (in our case). And like logistic regression SVM ends up with a separating hyperplane. However, criteria for constructing the separating hyperplane are different. SVM aims to find a linear separation with maximum margin – the distance between the recognized classes. Figure 8 provides an illustration. The axes correspond to different context features (in a way similar to section 2.1.5). Line (for more dimensions – plane or hyperplane) separates the cases of situation

occurrence and non-occurrence. Here occurrence is claimed if a context state is on top-right of the separating line. Labeled data, used to learn the separating line position, are marked as **o** (situation occurred) and **x** (situation did not occur). Margin is the area between two classes (in the figure 8 – between two grey lines on both sides of separating line, those lines are parallel to the separating line). The distance from the separating hyperplane to each class is denoted as *d*. The hyperplane is in the center of margin area, for both classes because it results in greater distance from separating hypeplane to the closest example point, and as a result in safer classification. The size of margin in figure 8 it is equal 2**d*. The goal of support vector machines method is to find the way of linear separation, which maximizes the margin. Refer to the tutorial [BU98] and book [RN09] for more details on support vector machines.

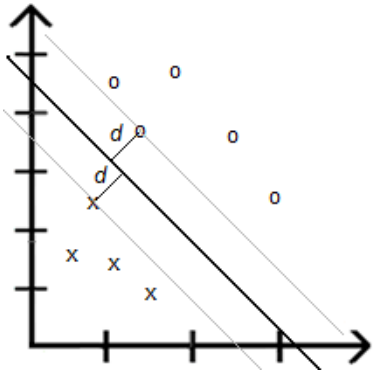


Fig. 8. SVM example for the case of two relevant context features.

Note that depending on how the hyperplane is drawn, different sample will be the closest ones. Finding separating hyperplane with the largest margin can be reduced to quadratic programming task, for which there exist multiple efficient algorithms. Some techniques are availablr for the case if labeled data are not completely linearly separable (see [BU98] for more details). Kernel trick allows extending SVM to non-linear cases and find non-linear separators. Refer to [BU98][RN09] for more information on kernel trick.

Support vector machines were used multiple times in context awareness and situation awareness. For example, Kanda et al. [KG08] used SVM to categorize the motion of customers in a smart shopping mall. The authors employed multiple laser finders to provide location readings. The recognized motion patterns included going straight, making right turn, making left turn, making U-turn, and stopping. Combined with other context awareness techniques, this information was used to deduce customer anticipations and proactively provide service to the customer.

In the paper [PR07] Patel et al. designed a system for activity recognition in a smart home home. SVM was used to classify the powerline transients, produced by home appliances. Therefore, the system was able to detect what appliances were turned on and off. In turn, this information was used to infer the activities of smart home inhabitants. The authors claim that they achieved 85-90% accuracy in detecting electrical events in a smart home.

Lee et al. [LL12] also used SVM to detect the appliances which smart home inhabitant uses. This information, combined with activity recognition, allowed detecting non-essential appliances, which do not participate in user activities and can be turned off. As an input the system used information from non-intrusive power meter. The accuracy of SVM recognition was around 88%. However, precision and recall were just around 52% and 43% respectively.

Next section discusses using neural netowrks for situation awareness in pervasive computing systems.

2.2.6 Using Neural Networks for Situation Inference

Neural networks are formal mathematical models that imitate biological neural structures. Starting back in the 1940s with the first models of neuron, it became one of the most popular ways of solving artificial intelligence related tasks. Learning capability allows neural networks to solve a variety of problems including pattern association, pattern recognition, function approximation.

Neural network can be defined as "a machine that is designed to model the way in which the brain performs particular task or function of interest" [Ha09]. A comprehensive list of neural network benefits is presented in the guide [Ha09]. The benefits relevant for pervasive computing task are following.

1. Nonlinearity. Neural networks can represent the situations that depend non-linearly from context features.
2. Adaptivity. Neural networks are well suited for supervised learning techniques. [Ha09].
3. Evidential response. Single neural network can be used to reason about several situations, whether the situations are mutually exclusive or not. Neural network both infer the ongoing situations and provide the confidence in the decision.

For comprehensive neural networks overview refer, for example, to the work by Russell and Norvig [RN09] or to the book [Ha09].

Neural network house [MD95] is one of the earliest examples of context aware smart home environment. The authors proposed ACHE system (Adaptive Control for Home Environments). ACHE aims to control all the aspects of comfort in smart home: ventilation, lighting, air and water temperature. Neural networks were the core of the algorithms for inferring inhabitants' lifestyle and for controlling smart home resources. The used sensors included light sensors, room temperature, sound level sensors, motion detectors, sensors for statuses of all doors and windows, illuminance sensors and many more. ACHE employed some situation awareness aspects as well. For example, it implemented occupancy model – for each zone (where each zone usually corresponded to a particular room) the system recognized, whether it is occupied by the user or not.

In the paper [KW10] authors proposed an activity recognition system that inferred user activities using cell phone accelerometers. The system extracted a set of features out of raw accelerometer data, and compared the performance of different activity recognition alogirthm on top of those features. The compared options were logistic regression, decision tree and multilayer perceptron, which is a very commonly used type of a neural network. The accuracy of multilayer perceptron reached 44% and 61% for the activities "going upstairs" and "going downstairs" respectively, but for all the other activities it exceeded 90% (activity "jogging" was recognized with over 98% accuracy).

Favela et. al [FT07] used neural networks for activity recognition in context aware hospital applications. The authors tracked the activities of hospital staff, and the recognized activities included patient care, information management, clinical case assessment. Four contextual variables were used as neural network input: location, artifcats, role and time. The feedforward neural network with 16 hidden units was trained using backpropagation algorithm. Overall, the authors achieved 75% activity recognition accuracy.

Neural networks are robust and versatile tools for learning various dependencies. However, neural networks contain some disadvantages as well. The main disadvantage of neural network is that it is a blackbox. The methods like decision trees, naïve Bayesian approach or logistic regression enable clear representation of situation formula. Human expert can interpret the learned dependencies and find out how exactly the situation is inferred. Neural networks are much less prone to human analysis.

Next section discusses the use of decision trees for situation awareness in pervasive computing. As opposed to neural netowrks, decision trees can be easily read and understood by the human expert.

2.2.7 Decision Trees for Situation Awarenes

Decision tree is a tree-like structure, which is used in decision support. Decision tree represents a function that uses a vector of attributes as an input and produces single output value – a decision. A simple example can be found in Figure 9. The figure shows a decision tree, which decides whether to give a loan to the customer. The input includes several facts about the customer: whether he/she is employed, the customer's salary and wehther customer has some assets. The graph in figure 9 is a tree graph. Decision making starts from the root node and proceeds on different directions depending on the input. For example, if the customer is employed, has high salary and asks for loan, on the first step decision making will take "Employed - Yes" direction from the root node and it will end up in the node "Salary". On the next step decision making will take "Salary - High" direction and end up in a decision node "Loan: Accept". It is a terminal node, and accepting the loan request is a final decision. Therefore, every node in a decision tree corresponds to an input feature to test, and the outcomes of testing determine, which child node should be taken next. For more details on decision trees and decision tree learning refer to [RN09].

Hong et al., [HS09] provided several reasons to use decision trees: they are easily understandable; they are capable of processing non-linear interactions among variables; they have very low sensitivity for the outliers; they can handle large amounts of data; and they can process both categorical and numerical data. Therefore, decision trees made their way into context awareness and situation awareness systems. In context awareness and situation awareness the input vector is a vector of context features, and the final decision is an occuring situation.

Kwapisz et al. [KW10], among other methods, used decision trees to recognize activity of a user from cell phone accelerometer readings. The activities included walking, standing, sitting, running, ascending and descending stairs. The authors designed 43 features of accelerometer readings, and those features were input of a decision tree. The accuracy of decision trees heavily depended on activity and varied from 55% (for going downstairs) to 96% (for jogging).

Bao and Intille [BI04] investigated several approaches to recognize user locomotion out of aceelerometer data. One of the inverstigated approaches was decision trees. Sensor readings from five wearable accelerometers gave enough information to recognize twenty locomotion activities. The recognized activities included walking, running, vaccuming,

standing still, climbing stairs, strength training. The authors claimed that decision trees have shown the best performance (compared to naiva Bayesian approach, instance-based learning and decision tables) and achieved around 84% accuracy.

In the paper [LO11] authors compared multiple approaches to human activity recognition on the dataset from the work [KA08]. The task was to infer user activities using the readings of many relatively cheap state change sensors. Sensors were attached to doors, furniture and appliances, and those sensors allowed to detect when an object is used by the user. Decision trees were one of the considered activity recognition approaches. Decision trees gained over 90% precision and recall.

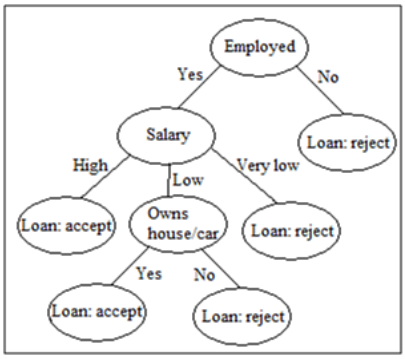


Fig. 9. Decision tree example.

It concludes the overview of the methods, which allow extracting situations out of labeled data. Next section discusses the methods to extract situation definitions out of unlabeled data. Those methods should overcome not only the challenge of extraction itself, but also the challenge of labeling the newly acquired situation.

2.3 Extracting Situations from Unlabeled Data

In previous sections we viewed manually defining the situations and learning the situations from labeled data. Manual definition of situation requires human expert to define the formula. Learning situation models from labeled data requires collecting training information, and then the formula can be obtained using supervised learning methods.

Learning situations from unlabeled data is mostly (but not exclusively [Ma04a][SL09]) used in location awareness. For example, location aware systems might need situations corresponding to the places that user frequently visits. Those situations can be *AtHome*, *AtWork* or *VisitingFriend*. However, the boundaries of the places are unknown in advance. The places like *AtHome* or *AtWork* are different for every user, so training sets collected in the lab are of no use for those situations. Requiring user to specify the information like home place, work place, locations of friends, places of interest in the city. is very intrusive and impractical. The same applies to creating labeled training set by periodically asking the user where he/she is.

A feasible solution is to find the clusters of location information and define the situations on that basis. Clusters of location readings are likely to correspond to places where user spends significant time, i.e. places of interest to the user. However, naming the newly acquired situation might be very challenging.

Learning situations from unlabeled data has the following advantages comparing to manual definition of situations and supervised learning:

- Situations can be learned at runtime. It is suitable for the cases when the definitions of situations heavily depend on the user, and prior training is impractical.
- The list of situations is not required. In previously mentioned examples the list of situations was pre-defined, and the main goal was to find the formulas, which transformed context into situations. However, in some situation aware systems even the number of situations is unknown in advance. For example, in context aware system that learns the hobbies of the user, or in location aware system that learns places of interest to the user there is no way to tell in advance, how many distinct hobbies will be detected, or how many places does the user often visit. For those cases unsupervised learning is a very efficient option.

Despite those advantages, the concept of learning situations from labeled data is used less frequently then the other methods of defining the situations. There are numerous challenges related to unsupervised learning of situation definitions, and those challenges make the application of unsupervised learning complicated. The main challenges are following:

- Possible slow start. The method requires multiple sensor readings sometimes over long time in order to identify the clusters.
- Clustering takes time and computational resources.
- Labeling the situation is additional challenge. Cluster of sensor readings might correspond to some situation of interest, but to what situation do they correspond? For example, it requires additional analysis to determine whether newly acquired cluster of location information is user's home, user's workplace or a shop that user often visits. Two possible options are asking the user and trying to label automatically.
- Distance metrics can be unclear. Clustering methods usually rely on the metrics of distance between sensor readings. For location aware systems the metrics is quite clear – it is real distance between locations. However, the distance metrics is unclear for two sets of sensor readings containing, for example, statuses of household appliances and wearable accelerometers on the user.
- Unknown number of situation restricts the scope of learning methods. Some clustering methods require the number of clusters as input (for example, K-means [WH07]), while other methods do not rely on that information (for example, DBSCAN [EK96] or growing neural gas [Fr95]). In some systems, like location awareness example in this section, the number of future situations is unknown, and it significantly restricts the scope of learning methods.
- Some sensor readings do not belong to any situation, and it can also restrict the scope of clustering methods. For example, location aware system can use clustering to identify multiple places of interest. However, some location measurements will correspond to the user moving from onew important location to another.
- It might be challenging to produce definitions of situations, which are suitable for runtime inference. Some clustering algorithms (like DBSCAN [EK96]) attribute all the points from the training set to some cluster or identify it as noise. However, when a new point arrives, it can be challenging to determine to what cluster it belongs.

The identified challenges are solvable for many practical applications, but the exact way to overcome the challenge depends on particular task.

Mayrhofer [Ma04a] performed a comprehensive work on context prediction, and addressed situation awareness questions as part of that work. In [Ma04a] the author viewed context prediction problem as situation prediction, and the situations were identified as

clusters in the space of context features. Mayrhofer compared multiple methods to cluster the sensor data and infer situations. The survey in [Ma04a] contained many clustering methods, out of which three were chosen for final implementation and evaluation in context prediction system: Kohonen's self-organizing maps [Ha09], K-means [WH07] and growing neural gas [Fr95][Ha01]. Distance metrics was Euclidean distance in a multidimensional space. Growing neural gas can learn the clusters of arbitrary shapes and it does not require knowing the number of clusters in advance (unlike K-means). The work [Ma04a] chose lifelong growing neural gas for situation recognition and namedtwo more reasons for that: growing neural gas is easy to use in online mode (unlike K-means and self-organizing maps) and also growing neural gas provides more stable cluster trajectories (i.e. much less jumps from cluster to cluster). After the situations are obtained by clustering, they are manually labeled and then used in context prediction effort. Mayrhofer's work will be addressed in more details when discussing context prediction.

Ashbrook et al. [AS02] employed the variation of K-means algorithm to cluster the GPS data and define and predict the location of the user. Original K-means algorithm has pre-defined number of clusters, which is ususally unsuitable for situation awareness approach. The version of K-means algorithm proposed in [AS02] is capable of handling variable number of clusters. For each cluster the algorithm marks all the points within a defined radius, and computes the mean of all the points. Then the system draws new radius from new mean, and repeats it until mean no longer changes. The algorithm [AS02] keeps adding the clusters untril there are no GPS points left. The system proposed in [AS02] also looks for sublocations within each location (i.e. subclusters within each cluster) in the same manner. The radius of clusters is determined by analyzing the dependency between the cluster radius and number of clusters. When the radius grows, the number of clusters decreases. This plot has a "knee", when the decreasing becomes slower. As Ashbrook et al. argue, at that point the number of cluster converges to the number of real meaningful location. Locations then can be labeled by the user; automated labeling is out of scope of [AS02]. Once locations and sublocations are found, the system uses Markov model [RN09] for location prediction.

Andrienko et al. [AA11] proposed the system, which extracts significant events and relevant places out of mobility data. Mobility data is aggregated from multiple users. On the first step the system extracted m-events (movement events) out of mobility information. Movement characteristics, which were used for event extraction, included speed, travelled distance, direction and temporal distances to the beginnings and ends of the trajectories. On the second step the system attempts to detmine place and time of likely event occurences, and the authors proposed two-stage clustering for that purpose. On the first stage the authors implemented spatio-temporal clustering of events – events are clustered according to their positions in space and time. The main purpose of the first stage is to remove the "noise" – occasional events that occur closely in space but at different time. The work [AA11] investigated the choice of distance function, which included space, time and thematic attributes of events. The considered options included distance on Earth for locations, distance in time for time properties (with some modigfications for cyclic properties like the day of the week) and Euclidean distance otherwise. The authors used density base clustering (DBSCAN [EK96] and OPTICS [AB99]), but defined the neighbourhood using the combination of spatial distance and distance thresholds for every considered attribute (note that for this approach single distance function is not required, and it overcomed one of the problems of clustering in situation awareness). On the second stage of clustering the system [AA11] applied spatial clustering, but only to the events that were not ruled out as noise on the first stage. Events then were further aggregated for subsequent analysis. The approach in [AA11] considered multiple possible user interventions for

choosing the attributes for events or for setting the distance thresholds. The collected and aggregated information can be used for multiple purposes. The authors provided examples of traffic congestion analysis and flight analysis.

Sometimes context awareness and situation awareness framework combine supervised learning and unsupervised learning in a single system. Siirtola et al. [SL09] combined supervised and non-supervised learning for activity recognition. The recognized activities included football, basketball, walking, running, Nordic walking, pedaling (spinning or cykling), gym training, roller skating, racket sports (tennis or badminton), floor ball and aerobics. The authors used the data from just one 2-dimensional wrist-worn accelerometer, which every user wore while perfoming the activities. The list of features used for clustering and classification included mean, variance, averga change between subsequent measurements. For clustering the authors used expectation-maximization algorithm [DL77]. The authors determined the correspondence between clusters and activities using the count of labeled points inside the clusters. C4.5 algorithm [Qu93] was used to construct decision tree, which takes a measurement as an input and attributes it to certain obtained cluster. Within the cluster another decision tree is constructed to determine the activity. As a result, the accuracy of activity recognition was increased from 80% for straightforward application of decision tree to 85% for the proposed combination of clustering and decision trees.

Van Kasteren et al. [KE11] proposed activity recognition mechanism that clusters sensor data into the clusters of actions, and then uses those clusters of actions to infer the activity. The final model is 2-layer hierarchical hidden Markov model. The model contains activities on the upper layer and sensor readings on the observation layer. Action cluster acts as an intermediate layer between raw sensor readings and activities. Timing dependencies are introduced between action clusters and activities. Refer to [KE11] for more details on the nodes and dependencies of the model. The authors claim that the proposed model outperforms straightforward hidden Markov model and hidden semi-Markov model.

Gordon et al. [GH12] proposed a system, which recognizes individual and group activities using the sensor data from smart coffee mugs and using mobile phone as a computational center. The auhors learned the activities out of labeled data and compared multiple algorithms for that purpose: K nearest neighbors, naïve Bayesian and decision trees (the book [RN09] contains an overview for all of those approaches). However, the authors used also unsupervised learning to aid the classification of activities. The authors compared several options of what to send from local activity recognition units to group activity recognition system. The options included raw sensor data, sensor signal features, local activity label and clustering results. The authors concluded that for now clustering-based approach results in sharp accuracy rate decrease (from 96% to 76%), but still it has some potential due to not requiring separate phase for local training and due to energy consumption reduction by 33%.

To summarize, learning the situations from unlabeled data poses multiple challenges, mostly related to labeling the clusters and defining the distance function. Still, unsupervised learning was used as situation inference method on multiple occasions [Ma04a][AS02][AH11][BZ12a][KB12]. In some cases supervised learning-based situation inference used unsupervised learning as an intermediate step [SL09][KE11][GH12]. Next section summarizes situation awareness techniques in pervasive computing, discusses the challenges of situation awareness and concludes chapter I.

3 Summary. Challenges of Situation Awareness in Pervasive Computing

Previous sections provided an overview of the methods used for situation awareness in pervasive computing area. The field of situation awareness contains multiple challenges. The challenges analyzed in this section are specific to entire group of approaches, or even to the whole field of situation awareness itself. The challenges outlined in this section are addressed later in the thesis.

The analysis of the overviewed situation awareness methods shows the following benefits and challenges of situation awareness approaches.

Definition of situations by human expert has the following benefits:

- It can be an efficient way to formalize the knowledge that the expert has. Sections 2.1.1-2.1.5 show a wide range of situation awareness applications. The situations were defined by experts in activity recogniton scenario [HI06], service adaptation scenario [CX05], smart home systesm [DP11] and many more. In each of the provided examples the developers already know the mapping between sensor data and situations, and the task mainly involves formalizing that knowledge properly.

- Manual definition of a situation allows representing situations in a clear and insightful manner. Sections 2.1.1-2.1.5 overview multiple methods of sitation awareness, and in all those methods the final situation can be read and understood by the expert. For example, the expert can understand the situation by reading logic formula or by visulaizing the graph of ontology.

However, manual definition of situation has also multiple challenges, which need to be addressed in order to use increase the approach efficiently.

- Manual definition has limited applicability. Developer can use the models described in sections 2.1.1-2.1.5 to define the situations manually. However, many other models (e.g. most of the models described in sections 2.2.1-2.2.6 and 2.3) are not suitable for manual definition.

- Manual defitinion is prone to human errors. The methods described in section 2.1.1-2.1.5 do not contain methods for automatically proving that the proposed situation definition matches expert knowledge.

- There migh be a tradeoff between complexity of development, complexity of reasoning and flexibility. When defining the situations, pervasive computing developer should take into account the following considerations:

1. Complexity of development. If the definition is complicated for a human expert to understand or compose, it can result in increased development efforts and in definition errors.
2. Flexibility. The model of a situation should be robust enough to infer real life situation out of sensor data.
3. Complexity of reasoning. At the runtime situation aware pervasive computing system constantly uses situation model to detect whether the situation occurs or not. If the model is too complex and the reasoning is too slow, it can significantly hamper situation awareness functionality and disrupt other functions of pervasive computing system.

Sometimes the aspects of development complexity, reasoning complexity and flexibility form a tradeoff, and this tradeoff is one of the challenges of situation awarenes. Chapter III of the thesis addresses this challenge, proposes a set of flexible situation models and analyzes the question of reasoning complexity.

As section 2.3 shows, unsupervised learning in situation awareness is primarily used in location awareness or in combination with supervised learning techniques. Many researchers decided to infer situations out of unlabled data for following resons:

- It is initially unclear how many situations are there going to be (e.g. how many importnat locations does the user visit). For example, refer to the article [AS02] or to the work [Ma04a] , both of which were overviewed in section 2.3.
- The definitions of each situation significantly depend on the user, and cannot be learned at the design time (e.g. important locations for every person are not the same). For example, refer to the articles [AS02] and [AA11].

One of the main challenges of unsupervised learning in situation awareness is the necessity of labeling. That challenge may result either in intrusiveness (if user performs the labeling [Ma04a]) or in additional efforts to design automated labeling system [BZ12a]. Sometimes if clustering is used as an intermediate step the clusters do not require labels and the challenge is avoided [KE11][GH12].

Chapters IV and V of the thesis address the challenges of learning situations in unsupervised manner and labeling those situations. The sections solve two different location awareness and activity recognition tasks by learning the situations from unlabeled data. Both sections address the challenge of proper clustering location information. Chapter IV proposes a method to automatically label the identified situations, while chapter V employs manual labeling and addresses the challenge of presenting the information to the user in a clear and meaningful manner in order to make labeling easier and more precise.

A challenge common for all proposed approaches is ensuring the correctness of situation definition. If a model of a situation is incorrect, it can lead to erroneous situation awareness results and, in turn, inadequate actions of pervasive computing system. Errors in situation model can be, for example, a result of human expert error or a result of overfit or underfit when learning the situation. Situation awareness functionlaity can be tested, but sometimes testing is not sufficient. Chapters VI and VII of the thesis propose, develop and evaluate situation verification – the technique to formally prove that the definition of situation is correct.

Chapters VIII-X address the challenge of context prediction, which is mostly represented by situation prediction techniques. Being able to predict future situations is of much use to pervasive computing systems.

The resolution of mentioned challenges will lead to more efficient situation awareness and, as a result, to significant improvements in pervasive computing.

Chapter II

ECSTRA – Distributed Context Reasoning Framework for Pervasive Computing Systems

Based on:

1. Boytsov, A. and Zaslavsky, A. ECSTRA: distributed context reasoning framework for pervasive computing systems. in Balandin, S., Koucheryavy, Y. and Hu H. eds. *Proceedings of the 11th international conference and 4th international con ference on Smart spaces and next generation wired/wireless networking (NEW2AN'11/ruSMART'11)*, Springer-Verlag, Berlin, Heidelberg, 1-13.

Foreword

This chapter presents ECSTRA – general purpose context awareness and situaiton awareness framework. ECSTRA provides a distributed context awareness and situation awareness engine, which can process both low-level and high-level context information. This chapter describes the foundations of ECSTRA, its architecture and implementation features.

ECSTRA framework was developed as part of PhD research project, and it provided a solid fundament for implementation and evaluation of the proposed algorithms and approaches. Most of the solutions proposed in the subsequent chapters, are implemented as extensions of ECSTRA framework, so ECSTRA is an important background for understanding the rest of the thesis.

Among other topics, this chapter describes the evaluation of ECSTRA framework. The goal of evaluation was to determine whether ECSTRA is suitable for real-time situation inference. In order to perform evaluation multiple different realistic situations and multiple different realistic context states were generated. The context attributes for evaluation were taken from common sense and from practice. For every generated situation the generated contribution functions were practically plausible (in terms of intervals and corresponding contribution values). Generated context states were given practically plausible values as well. Therefore, the evaluation provided some representation of how ECSTRA can work in practical scenario.

ECSTRA has proven its practical usefulness in smart home environment. In collaboration with INRIA, ECSTRA was incorporated as a context awareness and situation awareness tool in a smart home solution. The usefulness of ECSTRA was certified by INRIA (see appendix).

ECSTRA – Distributed Context Reasoning Framework for Pervasive Computing Systems

Abstract. Pervasive computing solutions are now being integrated into everyday life. Pervasive computing systems are deployed in homes, offices, hospitals, universities. In this work we present ECSTRA – Enhanced Context Spaces Theory-based Reasoning Architecture. ECSTRA is a context awareness and situation awareness framework that aims to provide a comprehensive solution to reason about the context from the level of sensor data to the high level situation awareness. Also ECSTRA aims to fully take into account the massively multiagent distributed nature of pervasive computing systems. In this work we discuss the architectural features of ECSTRA, situation awareness approach and collaborative context reasoning. We also address the questions of multi-agent coordination and efficient sharing of reasoning information. ECSTRA enhancements related to those problems are discussed. Evaluation of proposed features is also discussed.

Keywords: Context awareness, situation awareness, context spaces theory, multi-agent systems, distributed reasoning, collaborative reasoning.

1 Introduction

Pervasive computing paradigm focuses on availability and graceful integration of computing technologies. Pervasive computing systems, like smart homes or micromarketing applications, are being introduced into everyday life. Context awareness is one of the core challenges in pervasive computing, and that problem has received considerable attention of the research community.

The majority of pervasive computing systems are massively multiagent systems – they involve potentially large number of sensors, actuators, processing devices and human-computer interaction.

In this paper we present ECSTRA - system architecture for multiagent collaborative context reasoning, and situation awareness. ECSTRA builds on context spaces approach [PL08b] as context awareness and situation awareness backbone of the system.

The paper is structured as follows. Section 2 discusses the related work. Section 3 briefly addresses context spaces theory, an approach that constitutes the basis for low-level context reasoning and situation awareness in ECSTRA. Section 4 describes the structure of ECSTRA framework and addresses each of its architectural components in details. Section 5 describes the implemented mechanism for collaborative context reasoning and context information dissemination in ECSTRA. Section 6 provides and analyzes evaluation results. Section 7 discusses the directions of future work and concludes the paper.

2 Related Work

The research community proposed various approaches to address the problems of context awareness, situation awareness and distribution of context information.

Padovitz et. al. in [PL08b] propose ECORA architecture. Being a predecessor to ECSTRA, ECORA utilizes situation awareness mechanism on the basis of context spaces approach. ECORA was a source of an inspiration for ECSTRA, but no code was reused. Comparing to ECORA, our approach has extended support for multiagent reasoning, enhanced support for sharing and re-using reasoning information and also natural support for context prediction and proactive adaptation integration [BZ09][BZ10b].

In the paper [KK05] authors introduced ACAI (Agent Context-Aware Infrastructure) system. ACAI approach introduced multiple types of cooperating agents: context management agent, coordinator agents, ontology agents, reasoner agents and knowledge base agents. Context is modeled using ontologies. Comparing to ACAI, our approach features less specialized agents that are less coupled with each other. Agents are acting and sharing information without establishing sophisticated hierarchy. It makes our approach more robust and flexible to common disturbing factors like agent migration or communication and equipment failures. Instead of using the ontologies like ACAI, ECSTRA uses the methods of context spaces theory, that provide integrated solution from low-level context quality evaluation to situation awareness.

The work [XP08] featured CDMS (Context Data Management System) framework. That approach introduced the concept of context space (the set of context parameters needed by context aware application) and physical space (the set of raw sensor data provided by environment). Dissemination of context data from physical spaces is arranged using the P2P network. The approach [XP08] provided very advanced solutions for context data retrieval: query evaluation, updates subscription, matching between context spaces elements and relevant physical spaces elements. Comparing to CDMS, our approach features much higher degree of independence between context aware agents. It ensures better capabilities of information exchange between peer reasoning agents, and it ensures the robustness to different agent entering or leaving the system. In our framework context reasoning is completely decentralized. Context aware agents are capable of exchanging the information between each other, not just from-bottom-to-top manner. We utilize relatively flat publish/subscribe system, which allows us to employ loose coupling between multiple context aware applications, and make the system even more robust to reasoning agent migration. Our approach to context reasoning is based on context spaces theory and situation awareness principle, which provides simple, but yet flexible and insightful way to reason about real life situations.

The work [SW11] proposed MADIP multiagent architecture to facilitate pervasive healthcare systems. As a backbone MADIP utilized secure JADE [BC11] agent framework in order to maintain scalability and security criteria. ECSTRA uses Elvin publish/subscribe protocol [E11], which provides sufficient functionality for the search and dissemination of necessary context information, and also ensures independence and loose coupling of reasoning agents. Exact mechanisms for context awareness and situation awareness are not in the focus of [SW11], while it was the main concern when developing ECSTRA.

3 Theory of Context Spaces

The context reasoning and situation reasoning mechanisms of ECSTRA framework are based on context spaces theory. The context spaces theory is an approach that represents the context as a point in a multidimensional space, and uses geometrical metaphors to ensure clear and insightful situation awareness. The main principles of context spaces theory are

presented in [PL04].

A domain of values of interest is referred to as a *context attribute*. For example, in smart home context attributes can be air temperature, illuminance level, air humidity. Context attributes can as well be non numerical, like on/off switch position or open/closed window or door.

In spatial representation a context attribute is an axis in multidimensional space. Multiple relevant context attributes form a multidimensional space, which is referred to as *context space* or *application space*.

The set of values of all relevant context attributes is referred to as a *context state*. So, the context state corresponds to a point in the multidimensional application space. Sensor uncertainty usually makes the point imprecise to a certain degree. Methods to represent context state uncertainty include context state confidence levels and Dempster-Shafer approach [Sh76].

The real life situations are represented using the concept of a *situation space*. Reasoning over the situation space converts context state into a numerical confidence level, which falls within [0;1] range. In context spaces approach a situation confidence value is viewed as a combination of contributions of multiple context attributes.

Confidence level can be determined using formula (1).

$$conf_S(X) = \sum_{i=1}^{N} w_i * contr_{Si}(x_i) \quad \textbf{(1)}$$

In formula (1) confidence level for situation S at context state X is defined by $conf_S(X)$. The context state vector X consists of context attribute values x_i, the importance weight of i-th context attribute is w_i (all the weights sum up to 1), the count of involved context attributes is N, and the contribution function of i-th context attribute into situation S is defined as $contr_{S,i}(x_i)$.

Contribution function is often a step function that is represented by formula (2).

$$contr(x) = \begin{cases} a_1, x \in (b_1, b_2] \\ a_2, x \in (b_2, b_3] \\ \dots \\ a_m, x \in (b_m, b_{m+1}] \end{cases} \quad \textbf{(2)}$$

Practically, any boundary can be included or excluded, for as long as the set of intervals covers entire set of possible values and the intervals do not overlap. For non-numeric context attributes the intervals are replaced with the sets of possible values.

Next section discusses our proposed ECSTRA framework.

4 ECSTRA Framework

ECSTRA (Enhanced Context Spaces Theory-based Reasoning Architecture) is a distributed mutiagent context awareness framework, with its architectural elements distributed across the pervasive computing system. ECSTRA is presented in figure 1.

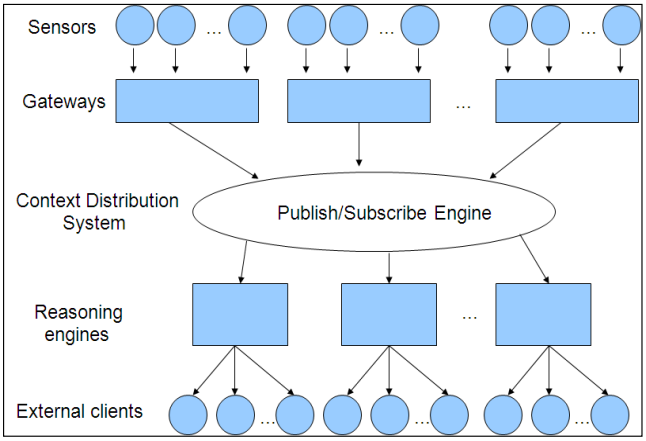


Fig. 1. Enhanced Context Spaces Theory-based Reasoning Architecture (ECSTRA).

Environmental parameters are measured and supplied by sensors. From the perspective of this research, human-computer interaction can be viewed as providing sensor input as well. However, raw sensor data is not the context information yet. Pervasive computing system needs to label that information, put necessary tags on it, evaluate its uncertainty and reliability, and distribute it within the system. These functions are carried out by the gateways. Gateways process information obtained through sensor networks, translate it into context attributes (most importantly, assign the unique names and define the uncertainties) and publish it to the special publish/subscribe service. Therefore, gateways process the sensor readings and create low-level context information out of it. Sensor uncertainty estimations can be either provided by sensors themselves or calculated by the gateway. Usually gateways are deployed on the devices directly connected to the sensor network base stations.

Context information is distributed using the publish/subscribe service. ECSTRA uses Avis open source implementation [A11] of Elvin publish/subscribe protocol [E11] for context information distribution. Reasoning agents subscribe to necessary context attributes information, and gateways publish the data they have. With separate publish/subscribe approach the context dissemination process becomes very flexible and almost transparent. Both the gateways and the reasoning agents can migrate freely, and it takes just resubscription in order to restore the proper information flow.

Reasoning engines are container entities that consist of one or more reasoning agents. The structure of reasoning engine and reasoning agents is depicted in figure 2.

Reasoning agents directly perform the context processing and situation reasoning. Every reasoning agent works with some part of the context. Sharing reasoning process between several reasoning agents can help to make reasoning agents more lightweight and parallelize the reasoning process.

Reasoning agent comprises context collector and application space.

Context collector block has the following functions:

1. Aggregating context data to a single context state. Sensor readings arrive one at a time, but application space requires the entire context state vector at any moment.

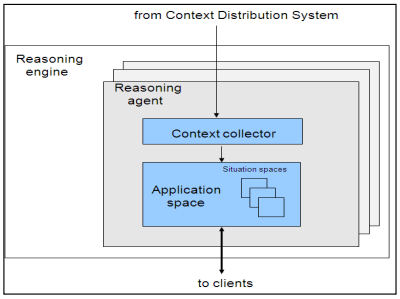


Fig. 2. Reasoning Engine Structure

2. Delivering new context states to the application spaces. Context collectors update the context state upon receiving of any new context attribute from publish/subscribe engine.

3. Managing the subscription to context information. It is the responsibility of context collector to subscribe to the necessary context information and to re-subscribe after the agent migration or restart.

4. Track the deterioration of quality of context over time. If there is no information received in a while, the context collectors update the context quality estimations and increase the expected uncertainty.

The application space block and the situation space blocks within it correspond to application space and situation spaces of context spaces theory. Application space handles context reasoning process and defines all the questions of situation algebra syntax and semantics. Situation space handles all the questions of situation representation. Currently ECSTRA supports original context spaces theory situation definition, fuzzy situation inference [DZ08], and a set of specially developed flexibility-optimized situation space formats, which are the subjects of ongoing work.

Another function of application space is sending notification to the clients that new context data have arrived. Application space operates only with context states as input data, and between the changes of context state the reasoning results remain the same. The client does not have to be subscribed to context data change, in order to request reasoning from application space. If the client is interested in receiving the context state itself right after the update, it can subscribe directly to the context collector using the application space to context collector interface.

As it was noted before, the context state within the application space does not change in the time between the notifications of context collector. That allows reducing the reasoning efforts by introducing the cache of reasoning results. If any ECSTRA client requested the reasoning about the certain situation, either like a single situation or within the situation algebra expression, the reasoning results are put into reasoning results cache. Later, if the context state did not change yet, but the same situation is requested again (once again, either as a single situation or a situation within algebra expression), ECSTRA takes the information from the cache. After context state changes, situation confidence levels might change as well, and cached results can no longer be trusted. Therefore reasoning results cache is cleaned when the new context state is received.

External clients are not a part of ECSTRA (some exceptions are provided in section 5). They connect to the application space, send reasoning requests and obtain the result of reasoning. For example, CALCHAS context prediction and proactive adaptation framework [BZ10b] can be a client of ECSTRA. Usually reasoning requests are presented in the format

of situation algebra expression, and reasoning results are either confidence level or Boolean value, that represents the validity of requested expression with respect to current context state.

As a part of this work, we implemented several enhancements to original ECSTRA architecture, which take the advantage of multi-agent and distributed nature of pervasive computing system.

5 Distributed Context Reasoning

The architecture of ECSTRA was designed to allow distributed reasoning and information sharing between agents. ECSTRA allows sharing high-level reasoning results. This approach encapsulates low-level reasoning efforts of different agents and reduces the amount of overlapped reasoning efforts between several reasoning agents.

Several particular features of ECSTRA enable distributed context reasoning. Those are context aware data retrieval, subscription-based reasoning result sharing and multilayer context preprocessing.

5.1 Context Aware Data Retrieval

Consider the following motivating scenario: the user is in a smart home. User's context is managed by PDA and the light level is requested from the sensor in the room. Practically it makes sense to treat the current light level around the user as a user's context attribute. However, the context attribute "CurrentLightLevel" will be represented by different sensors depending on the context. Here the required sensor will depend on user location.

To summarize, sometimes single context attribute corresponds to different sensors in different occasions. The context aware data retrieval aims to overcome that problem. The idea is to adaptively resubscribe to different context sources, depending on the current context information itself.

The method is to enhance collector with two additional functions: resubscription on request, and masking the global name of context attribute to replace it with its local name. As a result, for the application space the technique is completely transparent, and the computational core does not require any modifications. New block, subscription manager is introduced to manage the subscription switching.

Context aware data retrieval architecture is depicted in figure 3.

Context collector accepts commands from subscription manager. Subscription manager, in turn, contains the set of rules that define resubscription procedures. Subscription manager acts as a client to the application space.

The simplified protocol of resubscription decision making is depicted in figure 4.

The efficient use of that technique can allow to significantly reduce the number of context attributes under consideration, and to bring the number of involved context attributes down to the tractable numbers even for large-scale systems. Also it can significantly simplify the development of situation spaces.

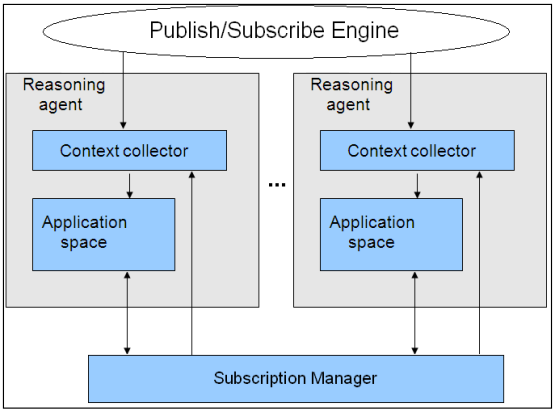


Fig. 3. Context Aware Data Retrieval – Architecture.

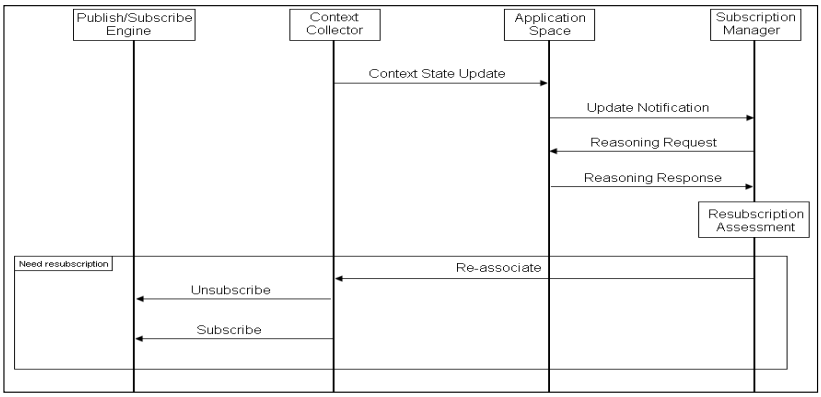


Fig. 4. Context Aware Data Retrieval – Protocol.

5.2 Reasoning Results Dissemination

In practice several remote clients can be interested in the results of reasoning agent work. Moreover, situation reasoning results can be taken as context attributes by other reasoning agents (if carried out properly, it can simplify the reasoning activities). The possible enhancement for that case is to reason about the situation and then to return the reasoning result into publish/subscribe engine. In ECSTRA it is implemented in a manner, presented in figure 5.

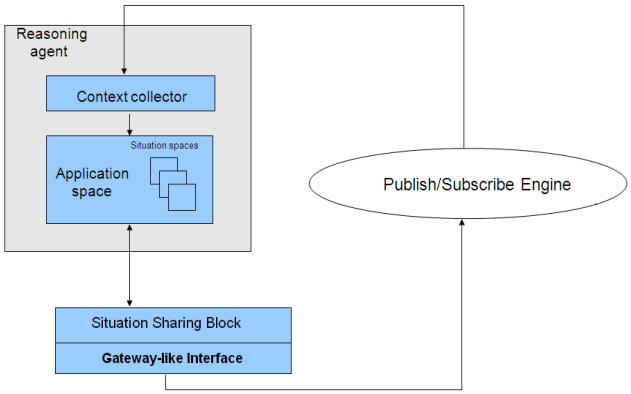


Fig. 5. Sharing of Reasoning Results.

As depicted in figure 5, situation sharing block returns the results of situation reasoning to publish/subscribe system. The subscribers to those results can be either other reasoning agents, or remote external clients themselves. The outer interface of situation sharing block resembles the interface of a gateway. It allows packing the shared information in context attribute-like format. This format allows managing context attributes and shared situations in a unified manner. The situation reasoning results (in the format of confidence level or binary occurrence) can be subscribed to and taken as an input by other reasoning engine, and this can create hierarchical distributed reasoning structure.

This approach can significantly reduce the necessary amount of reasoning activities and allow efficient sharing of information between reasoning agents and the external clients. Also this approach can naturally construct a hierarchy of reasoning activities, and this hierarchy will be relatively flexible and robust to agent migration and replacement, especially if combined with context aware data retrieval.

5.3 Multilayer Context Preprocessing

If both the number of context attributes and the number of reasoning agents are large, the efforts for context preprocessing might be significant. Context preprocessing efforts can be reduced by applying the multilayer context preprocessing technique, depicted in figure 6.

If N reasoning agents within reasoning engine have the common subset V of context state vector, the system can construct the context collector for context state V. Then obtained context state V can be used by all N context collectors directly. As a result, instead of N times preparing the vector V, that vector will be derived just once, and then distributed among all the interested reasoning agents.

So, multilayer context preprocessing approach can reduce the context preparation efforts. It is completely tolerant to the migration of context sources. However, multilayer structure can cause problems during the reasoning agents migration. As a result, multilayer context preprocessing should be used when there are many reasoning agents, but those reasoning agents are not likely to migrate separately.

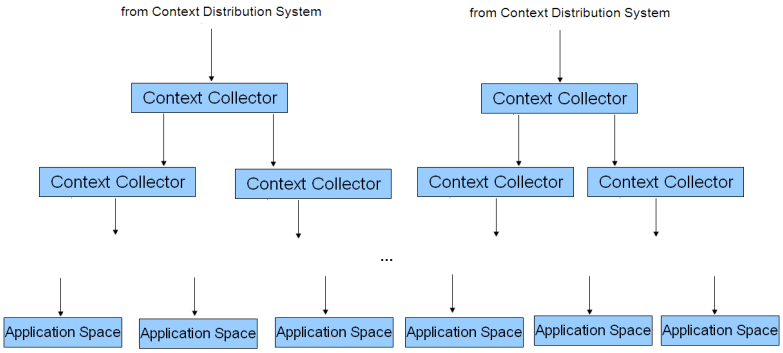


Fig. 6. Multilayer Context Preprocessing.

6 Evaluation of Situation Reasoning

The theoretical analysis of situation reasoning complexity is presented in table 1. The definition of the situation space is presented in section 3 in expressions (1) and (2). We refer to the total number of involved context attributes as N, to the number of involved intervals on i-th context attribute as m_i, and to the total number of involved intervals on all the context attributes as $M = \sum_{i=1}^{N} m_i$.

It should be noted that $N \geq M$ – there is at least one interval per context attribute. In practice often N>>M. Summarizing table 1, we expect that reasoning time will be linearly dependent on the number of intervals.

The experiment was performed as follows. Situation spaces and context states were randomly generated. There were 1000 randomly generated situations in a single application space. For every generated situation the number of intervals was generated uniformly between 1 and 60. Then the distribution of intervals between context attributes was generated uniformly. For every situation the reasoning was performed 10000 times without using the results cache, and then average reasoning time was taken as a result. Testing was performed on Lenovo ThinkVantage T61 laptop. The results are depicted in figure 7.

Table 1. Situation Reasoning Complexity

Operation	**Order**	**Details**
+	O(N)	On formula (1) the sum contains N summands. There are no more additions involved.
*	O(N)	On formula (1) every summand has one multiplication within. There are no more multiplications involved.
comparison	O(M)	Consider formula (2). For every context attribute the necessary interval will be taken the last in the worst case. That gives O(P) comparisons as the worst case estimation.

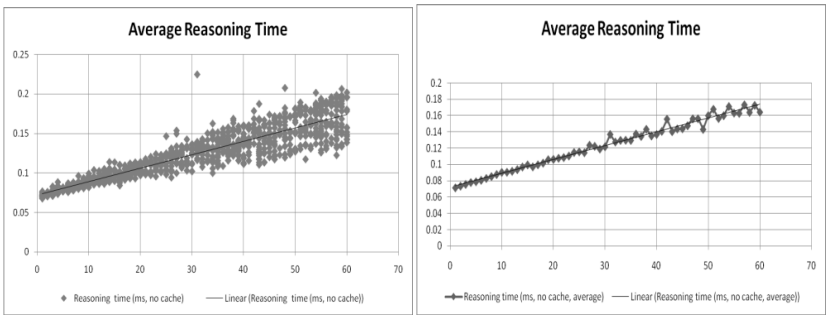


Fig. 7. Situation Reasoning Efficiency

The exact analysis of hetersocedactic properties of situation awareness complexity is being done as a part of advanced situation awareness, which is mentioned as future work direction in section 8. For the purpose of current research averaging out the results for every number of intervals (right plot in figure 7) provides enough accuracy.

Situation cache provides significant improvement in reasoning time, in comparison with straightforward situation reasoning. The experiment was performed in a similar manner on similar equipment. The reasoning time was calculated as an average for 1000 reasonings. The results are presented in figure 8 and table 2.

Table 2. Situation Cache Efficiency

Cache Size (situations)	25000	26000	27000	28000	29000	30000	31000
Average Reasoning Time (ms)	0.0653	0.0628	0.0623	0.0623	1.1836	1.1972	1.4400

As expected, if the reasoning result is taken from the cache, reasoning time does not depend on the size of situation space, but it does depend on the number of situations in the cache. As it is shown in table 2 and figure 8, until the amount of situations in the cache reach 29000, reasoning time is ~0.06 ms, which is less than time for reasoning about 1-interval situation. However, when the number of situations in the cache reaches 29000, the reasoning time starts to grow rapidly.

So the general recommendation is to use the situation cache even for the situations with low number of intervals, unless the count of situations in the cache exceeds 28000.

7 Conclusion and Future Work

In this work we presented the pervasive ECSTRA computing framework and application, that is capable of context reasoning and situation reasoning. ECSTRA is designed to fit multi-agent and highly distributed nature of pervasive computing systems.

We identified the following possible directions of future work:

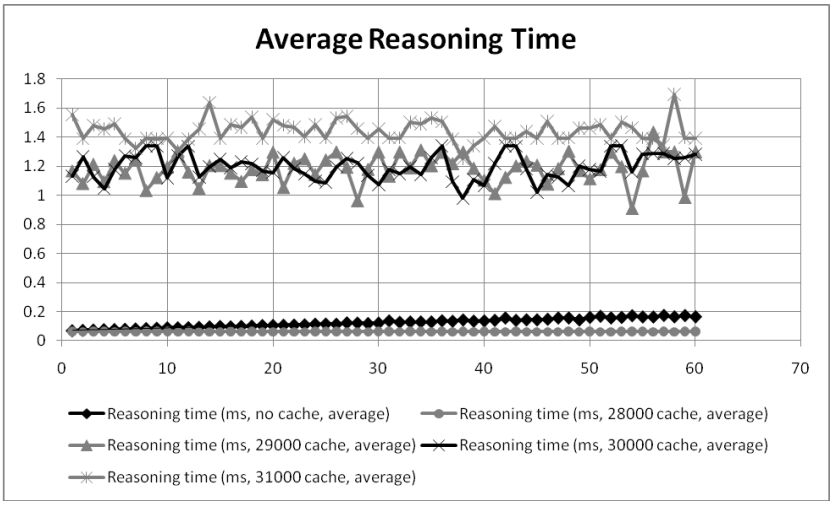


Fig. 8. Situation Cache Efficiency

1. **Advanced context aware data retrieval.** Currently the context aware resubscription technique is defined by static rules. It might work well for the relatively small systems, but for large-scale pervasive computing systems like smart towns it will result in enormous amount of rules. In order to address large-scale context aware data retrieval problem, we need advanced language of re-subscription rules, combined with efficient context attribute naming technique.
2. **Advanced situation awareness.** The situation reasoning techniques of context spaces theory provide a memory efficient and fast situation awareness solution, but sometimes they lack flexibility, and many real-life situations cannot be defined in the terms of expressions (1) and (2). The search of new situation definitions and analysis of its efficiency is the subject of ongoing work.
3. **Reliable distributed context delivery.** Reasoning agents are mostly vulnerable when they are migrating. If the context information update arrives during the migration of the agent (after unsubscribing, but before subscribing), it can as well be lost. In that case, loose coupling between sender and receiver, the important benefit of publish/subscribe system, becomes a disadvantage. The possible remedy for that problem is establishing some kind of knowledge storage agents that contain up-to-date data. Another possible option is introducing the request for data within publish/subscribe space.
4. **Smart situation cache.** Currently situation cache can be either on or off. If situation cache is on, it stores any situation reasoning results. In section 6 we proved that situation cache significantly reduces reasoning time, unless there are tens of thousands of situations in it. Situation cache can be further enhanced by smart decision making about whether to put the situation in it or not. Situations in the cache can be preempted depending on number of intervals (and, therefore, expected saved time). Another possible enhancement is to allow entire situation algebra expressions in the cache. The implementation details of those techniques, as well as efficiency of those methods, are yet to be determined.

Chapter III

From Sensory Data to Situation Awareness: Enhanced Context Spaces Theory Approach

Based on:

1. Boytsov, A. and Zaslavsky, A. From Sensory Data to Situation Awareness: Enhanced Context Spaces Theory Approach, in *Proceedings of IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC)*, 2011 , pp.207-214, 12-14 Dec. 2011. doi: 10.1109/DASC.2011.55.[5]

[5] The paper [BZ11b] has won the Best Paper Award of the Ninth International Conference on Pervasive Intelligence and Computing (PICom2011). PICom2011 and DASC2011 conferences have joint proceedings.

Foreword

This chapter addresses the research question 1 – how to derive a mapping between context information and ongoing situations? As Chapter I identified, one of the possible approaches is to define the mapping using expert knowledge. However, in order for that approach to be effective the situation models need to be flexible enough to represent real life situations, the situation models needs to be clear enough to be composed by human expert and the reasoning complexity of the situations should be suitable for real time situation inference.

This chapter proposes an enhancement of context spaces theory approach with new situation modeling techniques. New situation types provide a robust solution to represent broad class of real life situations. Moreover, otrthotope-based situation spaces provide a background for the verification approach, which is proposed in subsequent chapters as an answer to the research question 2. The proposed enhancements are implemented as an extension of ECSTRA framework, which was described in chapter II.

From Sensory Data to Situation Awareness: Enhanced Context Spaces Theory Approach

Abstract. High-level context awareness can be significantly improved by the recognition of real-life situations. The theory of context spaces is a context awareness approach that uses spatial metaphors to provide integrated mechanisms for both low-level and high-level context awareness and situation awareness. Taking context spaces theory situation awareness as a baseline, we propose and analyze the enhanced situation awareness techniques, which allow us to reason about broad class of real-life situations. We also improve reasoning about the relationships between situations, and discuss how it relates to newly proposed situation awareness approaches. Practical evaluation of the results is also discussed.

Keywords: context awareness, situation awareness, context spaces theory, pervasive computing.

1 Introduction

Context awareness is a key feature of pervasive, ubiquitous and ambient computing. For example, ambient intelligence systems (like smart homes or smart offices), social networks and micromarketing applications extensively utilize context awareness methods. High-level context awareness can be enhanced by situation awareness – the recognition of real-life situations.

Consider an example scenario. John works in the office at the construction site, and his workplace environment is at constant risk of problems: surrounding works can produce excessive noise, air might get dusty and polluted, power outages can lead to illuminance problems. In order to provision environmental conditions for his work, pervasive system needs to be aware of situations like "Light_Level_Insufficient", "Noise_Level_Too_High" or "Workplace_Environment_OK". If there are any problems, system should take corrective actions: for example, switch to backup power supplies, engage additional ventilation, close doors and windows to reduce noise. So, situation awareness is important enhancement of context awareness and backbone functionality for further decision making.

The situation from context awareness perspective can be defined as «*external semantic interpretation* of sensor data» [YD12]. The situation model is a method to represent a situation in a manner plausible for automated inference. The situation can be modeled as a cluster in a space of context features [Ma04a], as an entity in the ontology [DZ08][ES07][WZ04], as a conjunction of context properties [AN06], among other non-exhaustive definitions. The important features of a situation model include acceptable reasoning complexity, clarity and readability by the expert, and the flexibility to represent the wide class of real-life situations.

Context spaces theory (CST) [Pa06][PL04] is a context awareness approach that uses spatial metaphors to reason about context and situations. Using context spaces theory as a baseline, this paper proposes qualitative extension and novel situation awareness techniques that achieve flexibility, concise and clear situation representation and tractable reasoning complexity.

The paper is structured as follows. Section 2 describes the related work. Section 3 addresses the basics of context spaces theory, describes situation reasoning approach and derives the complexity evaluation for it. Section 4 provides the sample motivating scenario.

Section 5 proposes and analyzes the enhanced situation awareness approaches. Section 6 contains the practical evaluation of new situation awareness methods. Section 7 provides summary, further work directions and concludes the paper.

2 Related Work

Detecting real-life situations received considerable attention in context awareness research community.

The solutions presented in this paper are based on context spaces theory. The theory of context spaces was proposed by Padovitz et. al. [Pa06][PL04]. In context spaces approach the context information was viewed as a vector in multidimensional space of context attributes, and situations were viewed roughly as subspaces in that space. The paper by Delir et. al. [DZ08] proposes fuzzy set based extension to situation definitions for context spaces theory. Comparing to the original context spaces approach, we propose more powerful situation awareness techniques that address broader class of real-life situations and significantly enhance reasoning about the relationships between situations.

Anagnostopoulos et. al. [AN06] proposed the situation awareness technique that inferred the situation as the conjunction of Boolean context features. This approach resembles the CST method of confidence level calculation (see section 3). However, the situation awareness methods of CST work with confidence levels, and that provides more flexibility when working with real-life situations. Moreover, CST is capable of handling unequal importance of different context features and, using the results of this paper, can avoid the independent contribution assumption.

The papers [BI04][IS09][KK07][RA04] perform situation and activity inference using naïve Bayesian approach. Despite the seeming similarity, CST situation awareness and the Bayesian approach employ different semantics. The Bayesian approach assumes that situation either occurs or not, and estimates the probability of occurrence. Context spaces theory uses semantics of uncertainty (in particular, fuzzy logic [DZ07] and Dempster-Shafer [Pa06] approaches) and degree of occurrence.

Mayrhofer [Ma04a] viewed context as a vector in a multidimensional space of context features. Situations were represented as the clusters in that space. That approach enabled automated situation detection with clustering algorithms, so the method proposed in the work [Ma04a] is effective if initially the situations of interest are unknown. In addition, that solution works well if context prediction is involved. Comparing to [Ma04a], our concept of situation enables more clear and more concise situation definition, as well as simpler situation reasoning. Moreover, our approach features situation algebra, which allows us to reason about relationships between situations. Context spaces approach can also integrate context prediction and acting on predicted context [BZ09][BZ10b] (but context prediction is out of the scope of this paper).

Papers [DS07][ES07][WZ04] suggested ontology-based situation reasoning. Ontologies provide powerful solutions to represent the relationships between different situations. However, context ontologies usually do not address the level of raw sensory data, and therefore ontology-driven situation awareness requires additional complementary low-level reasoning. Comparing to ontology-based situation awareness our approach addresses all levels of context and features an integrated set of reasoning methods for both high-level context and low-level context.

3 The Theory of Context Spaces

The context spaces theory (CST) is an integrated approach for context awareness and situation awareness. CST uses spatial metaphors to achieve clear and insightful context representation. The foundations of context spaces theory are provided in the article by Padovitz et. al. [PL04]. In this section we define a set of related terms that will be used throughout the paper.

A domain of values of interest is referred to as *context attribute*. Context attributes can be either measured by sensors directly, or derived from sensory data. For example, air temperature, light level, noise level, air humidity can be the context attributes for a smart office.

Context attribute can be viewed as an axis. The exact value on the axis (e.g. particular air temperature at certain time or particular light level at certain time) is referred to as *context attribute value*.

An entire set of relevant context attributes constitute a multidimensional space. This space is referred to as *application space* or *context space*.

A set of all relevant context attribute values at a certain time is referred to as a *context state*. So, a context state represents a point in the context space. Context state point is usually imprecise due to sensor uncertainty.

Situation space is designed to represent real life situation. Reasoning about the situation in original context spaces theory worked in a following manner [PL04]. The input data for the reasoning process is the context state. The reasoning result is a confidence level – a value within the range [0;1] , that numerically represents the confidence that the situation is occurring. Confidence level can be calculated according to formula (1).

$$\text{conf}_S(X) = \sum_{i=1}^{N} w_i * \text{contr}_{S,i}(x_i) \quad (1)$$

In formula (1) $\text{conf}_S(X)$ is a confidence level for situation S at context state X, a particular context attribute within X is referred to as x_i, the importance weight of i-th context attribute is referred to as w_i (all the weights sum up to 1), the number of relevant context attributes is N, the contribution value of certain context attribute into total confidence value of the situation is referred to as $\text{contr}_{S,i}(x_i)$.

Contribution function is usually a step function over certain context attribute. It can be expressed by formula (2).

$$\text{contr}_{S,i}X = \begin{cases} a_1, x \in (b_1, b_2] \\ a_2, x \in (b_2, b_3] \\ \cdots \\ a_{K_i}, x \in \left(b_{K_i}, b_{K_i+1}\right] \\ a_{i,default}, otherwise \end{cases} \quad (2)$$

In formula (2) the values a_j are the contribution values, corresponding to certain interval. If the i-th context attribute value does not correspond to any interval, $a_{i,default}$ contribution is assigned. Contribution values are within the range [0;1]. The boundaries of the intervals $(b_j, b_{j+1}]$ can be either included or excluded, as long as the intervals do not overlap with each other. The total number of intervals for all context attributes from now and on will be referred to as $P = \sum_{i=1}^{N} K_i$.

So, according to formula (1), the original CST situation implies that the total situation confidence level comprises independent contributions of various context attribute values. Independent contributions of different context attributes can be a benefit from the perspective of reasoning complexity and memory consumption. However, the independence of contributions can result in significant lack of flexibility, especially for representing the relationships between the situations. We are going to address this problem in more details in sections 4 and 5.

As a part of this work, we analyzed the complexity of original CST situation reasoning. The results are depicted in table 1. If there exist at least one interval per context attribute, it means that $P \geq N$. In practice often P>>N, and therefore the expectation is to have O(P) reasoning time – linear dependency between reasoning time and number of intervals. Practical evaluation of that claim is provided in section 6.

In order to reason about situation relationships, original CST provides the following situation algebra operations.

1. AND: Confidence in the fact that all situations occur simultaneously.
2. OR: Confidence in the fact that at least one of the situations occurs.
3. NOT: Confidence in the fact that situation is not occurring.

Expressions (3) present the definitions of the operations.

$$\begin{aligned} &\text{AND: } \text{confA} \,\&\, B(\text{X}) = \min(\text{confA}(\text{X}), \text{confB}(\text{X})) \\ &\text{OR: } \text{confA} \mid \text{B}(\text{X}) = \max(\text{confA}(\text{X}), \text{confB}(\text{X})) \\ &\text{NOT: } \text{conf!}\,\text{A}(\text{X}) = 1 - \text{confA}(\text{X}) \end{aligned} \tag{3}$$

More complex situation algebra expressions can be calculated recursively, using the set of operations (3) as a basis.

4 CST Situation Awareness Challenges – Motivating Scenario

CST situation representation provides a set of tools, useful for many practical situation awareness cases. However, when the situation relationships are involved, the capability of original CST situation definition might be insufficient.

Consider a sample scenario – a smart office that monitors the workplace environment. Smart office has a light sensor and a sound sensor deployed. Each of those sensors has a directly corresponding context attribute: respectively LightLevel (measured in lx) and NoiseLevel (measured in dB).

Consider two CST situations: *LightLevelOK* and *NoiseLevelOK*. They define respectively whether the workplace has sufficient illuminance and whether the noise level at the workplace is acceptable. Expressions (4) and (5) represent situations *LightLevelOK* and *NoiseLevelOK*.

$$\text{LightLevelOK} = \begin{cases} 0, & \text{LightLevel} < 350 \\ 0.5, & \text{LightLevel} \in [350, 500) \\ 1, & \text{otherwise} \end{cases} \tag{4}$$

$$\text{NoiseLevelOK} = \begin{cases} 1, & \text{NoiseLevel} \leq 40 \\ 0.7, & \text{NoiseLevel} \in (40, 50] \\ 0.3, & \text{NoiseLevel} \in (50, 60] \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

Table 1. Original CST Situation Reasoning Complexity

Operation	Order	Explanation
+	O(N)	Sum in formula (1) has N summands.
*	None	Sum in formula (1) has N summands. Every summand has 1 multiplication operation. However, if the weights are multiplied by corresponding contribution levels in advance, there is no need for multiplication at all.
comparison	O(P)	Consider formula (2). In the worst case K_i comparisons will be required to find the contribution level. For N context attributes the number of comparisons is $\sum_{i=1}^{N} K_i$=P.
memory	O(P)	For every context attribute situation needs to store K_i contribution values and K_i+1 interval borders and inclusion levels per every axis. That gives O($\sum_{i=1}^{N} K_i$) = O(P) memory consumption. The situation also needs to store N weights, but if they are applied in advance, no additional memory is needed.

A compound situation *ConditionsAcceptable* determines whether the workplace has acceptable environmental conditions for the office worker. The proposed example is simplified, so in this scenario *ConditionsAcceptable* comprises only illuminance level and noise level. We define *ConditionsAcceptable* as *LightLevelOK & NoiseLevelOK*, where AND operation is performed according to the rules of CST situation algebra, presented in formulas (3).

The construction of *ConditionsAcceptable* situation is depicted on figure 1.

In order to derive *ConditionsAcceptable*, situation algebra was applied to expressions (4) in a straightforward manner. The resulting situation *ConditionsAcceptable* is depicted in figure 2. In a formal way *ConditionsAcceptable* situation can be defined according to formula (6).

$$\text{ConditionsAcceptable} == \begin{bmatrix} 1,(\text{LightLevel} \geq 500) \wedge (\text{NoiseLevel} \leq 40) \\ 0.7,(\text{LightLevel} \geq 500) \wedge (\text{NoiseLevel} \in [40,50)) \\ 0.5,(\text{LightLevel} \in [350,500)) \wedge (\text{NoiseLevel} \leq 50) \\ 0.3,(\text{LightLevel} \geq 350) \wedge (\text{NoiseLevel} \in [50,60)) \\ 0,(\text{LightLevel}<350) \vee (\text{NoiseLevel}>60) \end{bmatrix} \quad (6)$$

So, *ConditionsAcceptable* can be viewed as real life situation from common sense point of view. Moreover, situation *ConditionsAcceptable* is a result of a simple situation algebra expression over original CST situations. But the distribution of confidence levels, provided in formula (6), is unrepresentable in terms of the original CST situation definition.

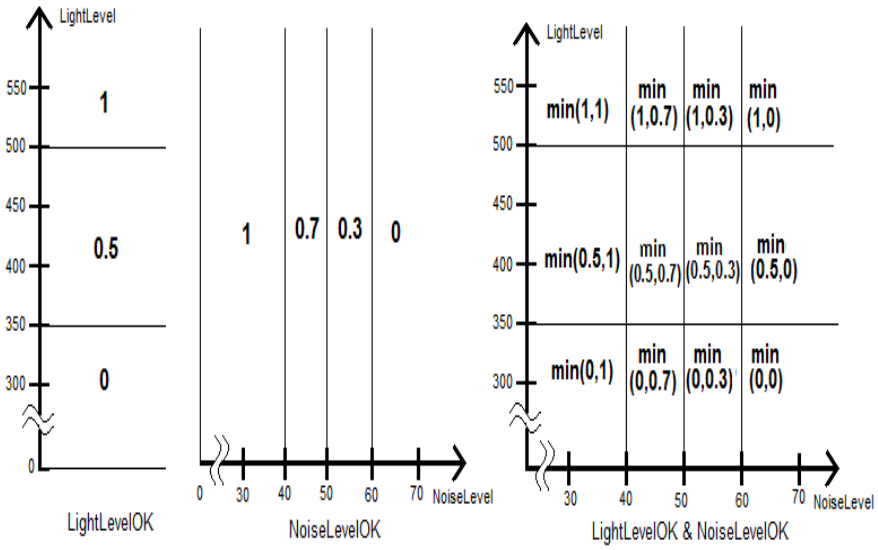


Fig. 1. Constructing *ConditionsAcceptable* situation

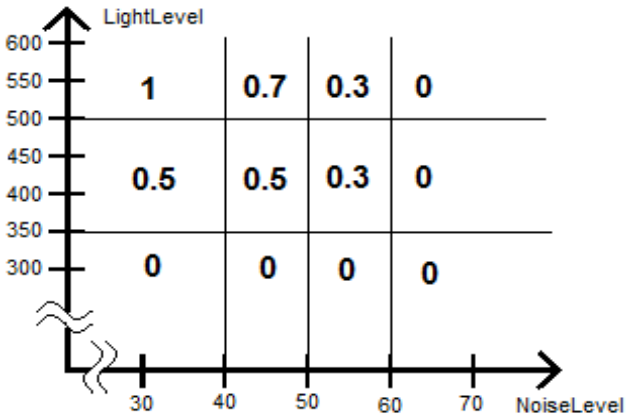


Fig. 2. *ConditionsAcceptable* situation

For the reasons of memory efficiency and reasoning complexity in original CST situations every context attribute contributes independently to the total confidence level. However, sometimes this assumption is too restrictive, especially if situation algebra is involved. For example, in this scenario *LightLevel* has zero contribution to *ConditionsAcceptable* if noise level is high and non-zero contribution otherwise.

We are going to refer to that sample scenario throughout the paper.

5 Enhanced Situation Representation

In order to make situation reasoning faster and cover the broader range of possible situations, we propose additional types of situation representation.

Dense orthotope-based situation space. Consider the situation *ConditionsAcceptable* from the example scenario presented in section 4. The distribution of confidence levels for *ConditionsAcceptable* is depicted in figure 2. The structure presented in figure 2 can be straightforwardly formalized into formula (7).

$$\text{ConditionsAcceptable} = \begin{bmatrix} 0, (\text{LightLevel} < 350) \wedge (\text{NoiseLevel} \leq 40) \\ 0, (\text{LightLevel} < 350) \wedge (\text{NoiseLevel} \in [40,50)) \\ 0, (\text{LightLevel} < 350) \wedge (\text{NoiseLevel} \in [50,60)) \\ 0, (\text{LightLevel} < 350) \wedge (\text{NoiseLevel} > 60) \\ 0.5, (\text{LightLevel} \in [350,500)) \wedge (\text{NoiseLevel} \leq 40) \\ 0.5, (\text{LightLevel} \in [350,500)) \wedge (\text{NoiseLevel} \in [40,50)) \\ 0.3, (\text{LightLevel} \in [350,500)) \wedge (\text{NoiseLevel} \in [50,60)) \\ 0, (\text{LightLevel} \in [350,500)) \wedge (\text{NoiseLevel} > 60) \\ 1, (\text{LightLevel} \geq 500) \wedge (\text{NoiseLevel} \leq 40) \\ 0.7, (\text{LightLevel} \geq 500) \wedge (\text{NoiseLevel} \in [40,50)) \\ 0.3, (\text{LightLevel} \geq 500) \wedge (\text{NoiseLevel} \in [50,60)) \\ 0, (\text{LightLevel} \geq 500) \wedge (\text{NoiseLevel} > 60) \end{bmatrix} \quad (7)$$

Original CST situation space uses separate contribution levels for every interval of every context attribute. In order to achieve more flexibility, a separate confidence level can be defined for every combination of context attribute intervals. Every row of formula (7) is a Cartesian product of intervals, and thus defines an orthotope [Co73]. Orthotopes are the basis of situation awareness improvements proposed in this paper.

By definition an orthotope is a Cartesian product of intervals [Co73]. So, for example, one dimensional orthotope is a line segment, two dimensional orthotope is a rectangle, three dimensional orthotope is rectangular parallelepiped. The example orthotope is provided on figure 3.

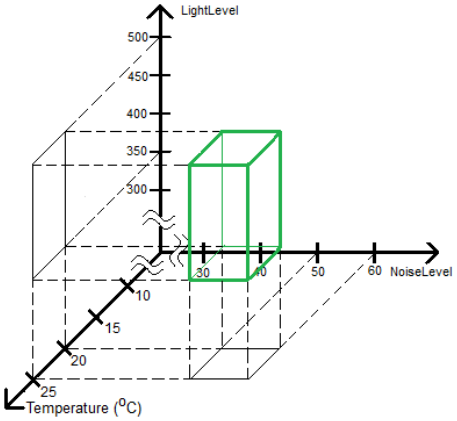


Fig. 3. An orthotope in the context space. It corresponds to intervals [20;25] on temperature axis, [50;60] on NoiseLevel axis and [350;500] on LightLevel axis.

In formula (7) the orthotopes densely cover (tesselate) the entire application space, so that any context state belongs to some orthotope. Therefore, this kind of situation representation is referred to as *dense orthotope-based situation space*.

Formal definition of generic dense orthotope-based situation space can be represented as follows. Consider that there are *N* context attributes involved in the situation. For the situation *ConditionsAcceptable* from the sample scenario, N=2 (*LightLevel* and *NoiseLevel*). Without the loss of generality, we can consider that the relevant context attributes correspond to positions *1...N* in the context state vector. Let *LightLevel* and *NoiseLevel* be the values number 1 and 2 in the vector respectively. The number of intervals, defined over *i-th* context attribute, is referred to as r_i. In the sample scenario, r_1 = 3 and r_2 = 4. The boundaries of i-th interval for j-th context attribute are referred to as $low_{j,i}$ and $high_{j,i}$. Every boundary of every interval can be either included or excluded, as long as every possible context state is included in one and only one orthotope. We define the total number of orthotopes as $L = \prod_{i=1}^{N} r_i$. For *ConditionsAcceptable* situation *L*=12. The total number of involved intervals is referred to as $R = \sum_{i=1}^{L} r_i$. For *ConditionsAcceptable* situation *R*=7.

Dense orthotope-based situation space is defined according to formula (8).

$$\text{conf}(X) = \begin{cases} a_1, (x_1 \in [low_{1,1}, high_{1,1}]) \wedge \ldots \wedge (x_N \in [low_{N,1}, high_{N,1}]) \\ a_2, (x_1 \in [low_{1,1}, high_{1,1}]) \wedge \ldots \wedge (x_N \in [low_{N,2}, high_{N,2}]) \\ \ldots \\ a_L, (x_1 \in [low_{1,r_1}, high_{1,r_1}]) \wedge \ldots \wedge (x_N \in [low_{N,r_N}, high_{N,r_N}]) \end{cases} \quad (8)$$

For every involved context attribute the set of intervals should cover the entire set of possible context attribute values. Also for every involved context attribute the intervals should not overlap with each other.

Table 2 presents reasoning complexity analysis for dense orthotope-based situation spaces. Table 2 shows that reasoning complexity is O(R). This claim is practically tested in section 6. Also table 2 shows that the major drawback of this situation representation is high memory consumption. In practice reasoning about orthotope-based situation space is done using decision trees. Pruning the decision tree is a way to improve both memory consumption and reasoning time. The exact potential benefit of decision tree pruning is a subject of future work.

Table 2. Reasoning over Dense Orthotope-based Situation Spaces

Operation	Order	Explanation
comparison	O(R)	In the worst case the proper interval will be encountered the last for every axis. In that case r_i interval inclusion tests will be performed for every context attribute, and it will result in $\sum_{i=1}^{L} r_i = R$ total comparisons.
memory	O(L+R)	Confidence level for very cell needs to be stored, as well as all the boundaries.

In order to improve memory consumption while retaining the flexibility, we developed another kind of situation representation.

Sparse orthotope-based situation space. Consider the situation *ConditionsAcceptable*, defined in the sample scenario in section 4. Formula (7) was derived by straightforward formalization of figure 2. But it is clearly visible that formula (7) is redundant, and formula (6) represents *ConditionsAcceptable* situation in a much more concise manner. Formula (6) can be derived from formula (7) by merging the neighboring orthotopes, if those orthotopes have the same associated confidence level. Situation algebra operators, presented in formulas (3), make it likely that the adjacent orthotopes will share the confidence level.

Formula (6) can be even further simplified, and the situation *ConditionsAcceptable* can be defined according to formula (9) or figure 4.

$$\text{ConditionsAcceptable} = \begin{cases} 1, \text{LightLevel} \geq 500 \wedge \text{NoiseLevel} \leq 40 \\ 0.7, \text{LightLevel} \geq 500 \wedge \text{NoiseLevel} \in [40,50) \\ 0.5, \text{LightLevel} \in [350,500) \wedge \text{NoiseLevel} \leq 50 \\ 0.3, \text{LightLevel} \geq 350 \wedge \text{NoiseLevel} \in [50,60) \\ 0, \text{otherwise} \end{cases} \quad (9)$$

In formula (9) the entire situation space is defined as a set of orthotopes in the context space, and each orthotopes is assigned a confidence level. But in contrast with dense orthotope-based situation space, the orthotopes are sparsely scattered throughout the context space, and the default confidence level is associated with the context state that do not belong to any orthotope. This kind of situation representation is referred to as *sparse orthotope-based situation space*.

Generic sparse orthotope-based situation space can be formally defined as follows. Consider that situation space is defined over N context attributes. Without the loss of generality, we can consider that the relevant context attributes correspond to positions *1...N* in the context state vector. For the situation *ConditionsAcceptable*, similarly to dense orthotope-based situation representation, N=2 (*LightLevel* and *NoiseLevel*). Let *LightLevel* and *NoiseLevel* be the values number 1 and 2 in the context state vector respectively. The number of orthotopes is referred to as Q. For the situation *ConditionsAcceptable* Q=4. Every orthotope is defined over N context attributes and contains one interval for each context attribute. Let the boundaries of i-th orthotope for j-th context attribute be $low_{j,i}$ and $high_{j,i}$. Every boundary of every orthotope can be either included or excluded, as long as orthotopes do not overlap.

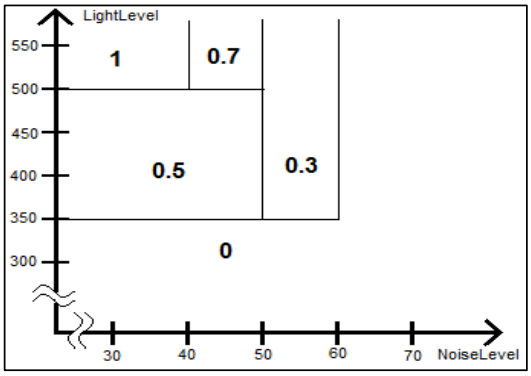


Fig. 4. *ConditionsAcceptable* situation – simplified

Sparse orthotope-based situation space can be defined according to formula (10).

$$\mathrm{conf}(X) = \begin{cases} a_{1,}(x_1 \in [\mathrm{low}_{1,1}, \mathrm{high}_{1,1}]) \wedge \ldots \wedge (x_N \in [\mathrm{low}_{1,N}, \mathrm{high}_{1,N}]) \\ a_{2,}(x_1 \in [\mathrm{low}_{2,1}, \mathrm{high}_{2,1}]) \wedge \ldots \wedge (x_N \in [\mathrm{low}_{2,N}, \mathrm{high}_{2,N}]) \\ \ldots \\ a_Q, (x_1 \in [\mathrm{low}_{Q,1}, \mathrm{high}_{Q,1}]) \wedge \ldots \wedge (x_N \in [\mathrm{low}_{Q,N}, \mathrm{high}_{Q,N}]) \\ a_{\mathrm{default}}, \mathrm{otherwise} \end{cases} \quad (10)$$

We performed complexity analysis for reasoning over sparse orthotope-based situation spaces. The results are given in table 3 and some necessary explanations are provided below.

Table 3. Reasoning over Sparse Orthotope-based Situation Spaces

Operation	Order	Explanation
comparison	O(Q*N)	At most N interval inclusion checks are required for each of Q subspaces.
memory	O(Q*N)	Situation space stores Q contribution levels and Q*N interval boundaries. The total order is O(Q*N).

Comparing to dense orthotope-based situation space, sparse orthotope-based situation space often represents situations in more clear and concise manner, and yet provides the same level of flexibility. Transitioning from dense to sparse orthotope-based situation space might improve memory consumption and reasoning time, but it depends on how many neighboring orthotopes share the same confidence level.

Situations of different types can be combined in the same application space and, moreover, different kinds of situation spaces can be combined in situation algebra expressions without altering the original concepts of CST situation algebra. Mixed situation spaces that have the features of original CST situation spaces on high level and dense orthotope-based or sparse orthotope-based situation spaces on low level are the subject of future work (see section 7).

Practical evaluation of different situation representation techniques is presented in section 6.

6 Reasoning Complexity Evaluation

The theoretical evaluation of situation inference complexity is presented in section 3 and section 5 (particularly, in table 1, table 2 and table 3). In this section we will address the practical aspects of situation reasoning.

Original situation space. Figure 5 shows testing results of ECSTRA reasoning for original context spaces theory situation space. Every point of the plot is the testing results for randomly generated situation. Abscissa contains the number of intervals for the situation (value *P*), and ordinate contains the average reasoning time in milliseconds. We generated 60000 random situations. For every situation the total number of intervals was chosen from [1;60] range uniformly. The distribution of intervals between context attributes was generated uniformly. The reasoning was performed at 1000 random context states for every situation. The result of every experiment is the average reasoning time.

The plot on figure 5 has visible heteroscedasticity, and it can obscure the results and mislead the analysis. The reason for heteroscedasticity is following: for any situation with *P* involved intervals, if *P=N* (one interval per axis) there are *P* inevitable interval inclusion

checks. It is the worst case for a situation. In the best case there are *P/2* interval inclusion checks in average: if *N*=1 the number of comparisons varies from 1 to *P* with average at *P*/2. So the expected lower border and upper border are linear, with the upper border around twice higher than the lower border. And that is visible on figure 5.

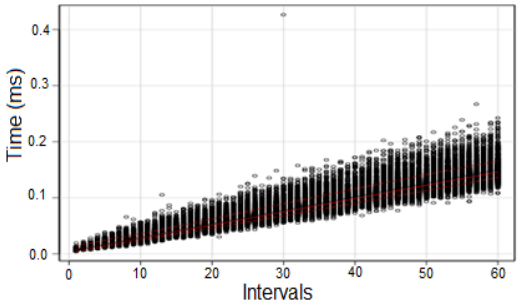


Fig. 5. Situation Reasoning Time – Original CST Definition

In order to have reliable estimations in presence of heteroscedasticity, we used the weighted regression technique. We used the method suggested, for example, in [Do07]. In addition we took the advantage of discrete explanatory variable, which allowed us to have variance estimations for every relevant point on abscissa. Regression analysis was performed using R [VS12] statistical software. The testing have shown that R^2 coefficient of weighted regression is equal to 96.18%, which shows good fit and practically proves the claims about linear algorithm complexity.

Dense orthotope-based situation space. The experiment settings for dense orthotope-based situation reasoning evaluation were similar to those for original CST situation reasoning evaluation. We generated 60000 random situations, where every situation contained up to 40 intervals.

The testing results are presented on figure 6. The abscissa contains *R* – the number of involved intervals, while the ordinate contains average reasoning time in milliseconds.

In order to prove linear trend, we performed regression analysis over testing results. To overcome heteroscedasticity weighted regression technique was used. R^2 coefficient is 0.87, and it shows good fit and practically proves linear dependency between reasoning time and total number of intervals.

Sparse orthotope-based situation space. Figure 7 contains evaluation results for reasoning over sparse orthotope-based situations. The experiment settings were similar to the experiments for evaluating original CST situation space and dense orthotope-based situation space. Test engine generated 60000 random situations with up to 60 intervals.

Figure 7 shows clearly visible heteroscedasticity. The reasons for heteroscedasticity are quite similar comparing to other experiments, but the features of situation representation introduce more variability in reasoning time. In order to analyze the data in presence of heteroscedasticity, we employed weighted regression technique. R^2 coefficient is 0.82, and it practically proves the linear trend.

To summarize, for all three mentioned situation representations, theoretical claims about reasoning complexity were proven practically. In addition, for all situation definitions the testing results showed heteroscedasticity: with growing explanatory variable, the variability of reasoning time grows as well. It makes reasoning time less predictable when the number of involved intervals increases.

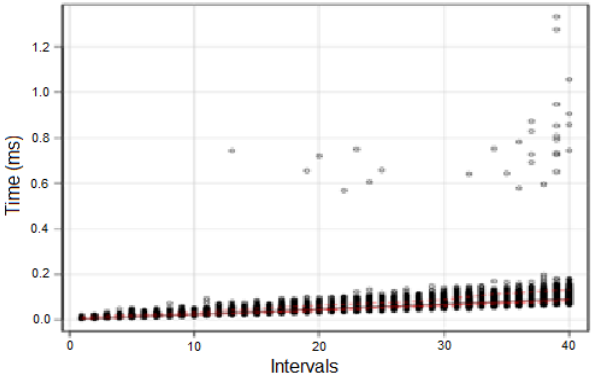


Fig. 6. Situation Reasoning Time - Dense Orthotope-based Situation Spaces

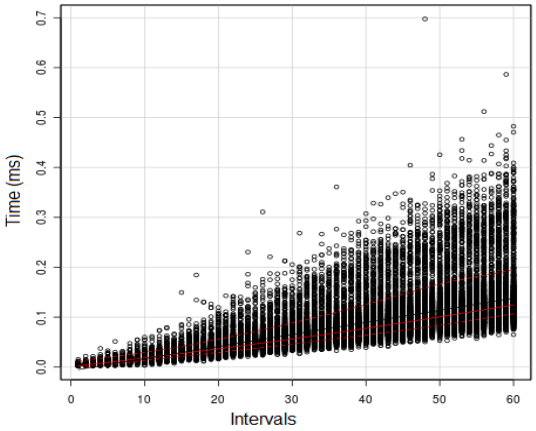


Fig. 7. Situation Reasoning Time - Sparse Orthotope-Based Situation Spaces

7 Summary and Future Work

In this paper we addressed the problem of situation awareness and made significant improvement to the situation awareness technique based on context spaces approach. Taking context spaces theory as a baseline, we developed enhanced situation awareness techniques that can address the broad class of real-life situations and reason about situation relationships in more efficient manner. The increasing flexibility of situation representation enables more versatile situation awareness, better generalization of context information and more intelligent decision making.

We consider the following directions of further work in situation awareness area:

1. **Mixed situation representation.** Situation space can be defined by combining the elements of original, sparse orthotope-based and dense orthotope-based situation spaces.

However, in order to construct mixed situation space, we need to identify which context attributes have mutually dependent contributions.

2. **Automated situation space definition.** Situations in CST are currently defined manually. This process can be cumbersome and prone to errors. Existing knowledge bases (e.g. ontologies of the subject area) might already have the necessary information to generate the situations, and extracting the situations from knowledge bases can eliminate the need for manual work.

3. **Run-time situation inference.** Situations of interest can as well be unclear during the system startup. Identifying the areas of context space that are likely to be the situations of interest is a subject of future work. For example, it can be achieved by clustering context states history.

4. **Situation awareness in absence of information.** Due to the sensor uncertainty and unreliability, the sensory data can become erroneous or missing. The goal of situation aware system is to retain as much situation awareness capability as possible in these circumstances.

5. **Context prediction and proactive adaptation.** Some papers addressed the problem of context prediction and acting on predicted context in context spaces theory [BZ09][BZ10b], but still there is a large room for improvements in the field. In particular, situation awareness advancements can enhance situation prediction area.

Chapter IV

Where Have You Been? Using Location Clustering and Context Awareness to Understand Places of Interest

Based on:

1. Boytsov, A., Zaslavsky, A. and Abdallah, Z. Where Have You Been? Using Location Clustering and Context Awareness to Understand Places of Interest. in Andreev, S., Balandin, S. and Koucheryavy, Y. eds. *Internet of Things, Smart Spaces, and Next Generation Networking*, vol. 7469, Springer Berlin / Heidelberg, 2012, pp. 51–62.

Foreword

Chapter I classified situation awareness approaches and identified a clear distinction between defining situations using expert knowledge and learning the situation definitions. This chapter answers the research question 1, but it takes another approach comparing to chapter III. While chapter III investigates defining the situations by hand, this chapter proposes an approach to learning the situations.

Most of situation awareness approaches require labeled data to learn the situation definitions, but this chapter proposes an approach to infer the situations out of unlabeled data. The solution proposed in this chapter chapter infers most important places that a user visits and autolabels the identified place, hence addressing both important problems of learning situations from unlabeled data. The proposed algorithms are implemented in ContReMAR application, which is based on ECSTRA framework.

Where Have You Been? Using Location Clustering and Context Awareness to Understand Places of Interest

Abstract. Mobile devices have access to multiple sources of location data, but at any particular time often only a fraction of the location information sources is available. Fusion of location information can provide reliable real-time location awareness on the mobile phone. In this paper we propose and evaluate a novel approach to detecting the places of interest based on density-based clustering. We address both extracting the information about relevant places from the combined location information, and detecting the visits to known places in the real time. In this paper we also propose and evaluate ContReMAR application – an application for mobile context and location awareness. We use Nokia MDC dataset to evaluate our findings, find the proper configuration of clustering algorithm and refine various aspects of place detection.

Keywords: context awareness, contextual reasoning, location awareness, sensor fusion.

1 Introducton

Present day mobile devices have access to multiple sources of location information. The possible sources include GPS sensors, WLAN location information, GSM localization, indoors positioning systems, dead reckoning. However, the sources of information are not always available. For example, GPS sensor is unstable indoors, it is battery consuming, and users usually turn it on only for driving from one place to another. In turn WLAN is often available at home and in office buildings, but it is rarely used outdoors. Therefore, for location awareness it is vital to fuse the location measurements from different sources. Location data fusion can remedy both with the unavailability of location information sources and the lack of precision of location information.

The area of mobile location awareness faces two major challenges that we are going to address in this paper:

- Detecting the places of interest out of location measurements that are fused from multiple sources.
- Finding the model of places of interest that allows feasible mobile real-time recognition using the measurements from multiple sources.

We used Nokia Mobile Data Challenge (MDC) dataset [LG12] as a benchmark, in order to evaluate location awareness algorithms, as well as to test the long-run performance of the application by imitating the sensor feed. We used ECSTRA toolkit [BZ11a] for context awareness and context information sharing.

The paper is structured in the following way. Section 2 proposes the approach to extract relevant places from location information. Section 3 discusses the architecture of ContReMAR – mobile context awareness application that we developed to evaluate the proposed approaches. ContReMAR has embedded capabilities for location awareness and location data fusion. Section 4 provides evaluation and demonstration of the proposed application. We use Nokia MDC data [LG12] in order to configure the parameters of clustering algorithms and increase the precision of place recognition. Section 5 discusses the related work. Section 6 summarizes the results, provides further work directions and concludes the paper.

2 Mobile Location Awareness

Location awareness is an important part of context awareness. The information about current or previously visited places can be used to provide timely assistance and recommendation for the user. The outcomes of location awareness can be also used as a baseline for activity recognition or context prediction. One of the main aspects of location awareness is detecting and labeling the places of interest.

In order to detect the relevant places, we need to identify the concept of a relevant place first. The place where the user has spent significant time is likely to be the place of interest for the user. The latter can be obtained by clustering the location measurements. We propose the following algorithm for extraction and identification of the relevant places. Evaluation of the proposed approach is provided in section 4.

Place Recognition Approach.

Step 1. Fusion of location measurements. We complemented the information obtained from GPS data with the location information obtained from WLAN. The GPS entries, which corresponded to high-speed movement, were removed from consideration.

Step 2. Cluster the location information. When logging is complete (e.g. at the end of the day), the application finds the clusters of location measurements to detect the places of interest. The nature of the task enforces following requirements for the clustering approach:

1. The user stays in the relevant place for significant time. So, the place of interest can be characterized by a relatively dense cluster of location measurements.
2. The number of places, that user has visited during a day, is not known in advance. Therefore, the number of clusters is initially unknown.
3. Some measurements do not correspond to any place of interest (e.g. user just walks on a street). Therefore, the clustering algorithms should not try to attribute all points to some cluster.

The approach, which satisfies all the constraints, is density-based clustering. DBSCAN [EK96] and OPTICS [AB99] are the most well-known algorithms of that approach. Those algorithms have the following parameters:

- *MinPts* – minimum number of points in vicinity, in order for the point to be in the core of the cluster.
- *Eps* – vicinity radius.

Step 3. Analysis of relevance. This step identifies, whether the detected clusters are the places of interest or not. For example, staying at the traffic light can result in multiple GPS measurements around the same spot, but it is not a place of interest. Irrelevant places can be filtered out by applying a threshold on the time spent at the place. More advanced aspects of relevance analysis are discussed in section 4.

Step 4. Auto-labeling the detected places. The detected place should be presented to the user in a meaningful way. Our approach combines two ways to auto-label the detected places.

Step 4.1. Obtain the list of possible relevant places. We use Google Places API [GP12] to detect the relevant places in the vicinity. The obtained list of places can be then filtered and ordered depending on at what time user was at that place.

Step 4.2. Analyze the time spent at a certain place of interest. In order to identify, what place of interest does the cluster correspond to, it is important to notice when the user was at that place. For example, if the user spent entire Wednesday at some place, it is very likely to be his/her work or study place, any night is very likely to be spent at home, and break in the middle of the working day is very likely to be lunch.

Step 5. Save the place description for later real-time recognition.

Section 3 discusses ContReMAR application, in which we implemented this location awareness approach. The evaluation of the proposed approach is presented in section 4.

3 ContReMAR Application

In order to prove, test and validate our approach, we designed and developed ContReMAR (CONText REasoning for Mobile Activity Recognition) application – a solution for mobile context awareness. The scope of ContReMAR is broader than just location awareness. However, in this paper we are going to focus on location awareness capabilities of ContReMAR, and its other aspects of mobile context awareness are out of scope of this paper.

The location awareness approach, proposed in section 2, was embedded into the ContReMAR application. The architectural and implementation solution are addressed in details in the subsections 3.1-3.3.

3.1 ContReMAR Architecture

The structure of ContReMAR application is depicted in figure 1. The blocks, which we designed and implemented specially for ContReMAR are depicted in green. The third-party solutions that we used are presented in yellow.

The application is divided into server part (which resides, for example, on a stationary computer or a laptop) and client part (which resides on the user device). Server side is responsible for computationally heavy and memory heavy parts of the work: logging the GPS data and identifying the clusters. The client side is responsible for real-time activity recognition. ECSTRA (Enhanced Context Spaces Theory Based Reasoning Architecture) framework [BZ11a], which is the basis of context reasoner, is available for Android platform. In order to evaluate the algorithms, we also used a simulation of mobile device, where the sensor feed was replaced with Nokia MDC [LG12] data flow.

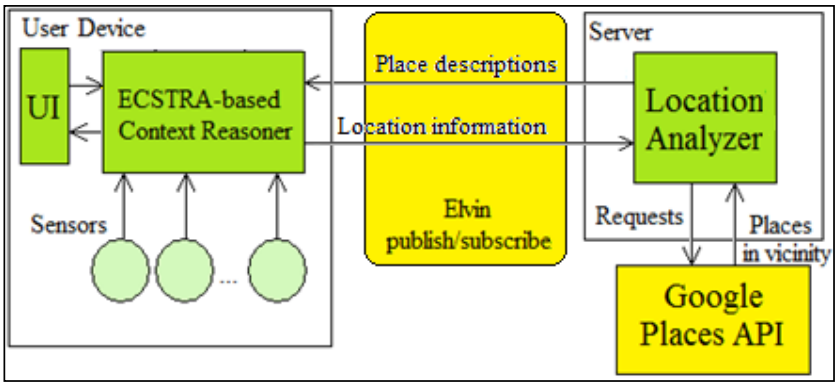


Fig. 1. ContReMAR Application Architecture

Figure 1 illustrates that ContReMAR application consists of the following components:

- **ECSTRA-based context reasoner.** We developed context reasoner for real-time location awareness, situation awareness and activity recognition. The detailed description of context reasoner component is provided in the section 3.2.
- **Location Analyzer.** We implemented location analyzer in order to infer new meaningful places from raw location data. Location analyzer is discussed in details in section 3.3.
- **User Interface.** It is a currently prototyped application component, which shows the results of context reasoning. The design of user-friendly interface is a subject for future work.
- **Elvin publish/subscribe environment.** In order to facilitate the communication between mobile device and the server side, we employed Elvin publish/subscribe protocol [E11]. Our application used Avis [A11] open source implementation of Elvin protocol. The proposed solution ensures seamless communication, even if both mobile device and server (which can be situated on a laptop) are moving between the coverage areas of different WLAN spots.
- **Sensors**. In non-simulated environment the sensors are merely the sensors situation on a mobile device. In order to evaluate our approach using Nokia MDC data [LG12] we also developed a simulated mobile device, which imitates sensor feed by substituting it with Nokia MDC data flow.

The application uses Google Places API [GP12] in order to detect the possible places of interest in the vicinity of the location.

3.2 Context Reasoner

We proposed and developed context reasoner for real-time inference of location, situations and activities. Context reasoner is based on ECSTRA framework [BZ11a], and has many of it components reused or extended from ECSTRA. ECSTRA is a general purpose context awareness and situation awareness framework, which acts a backbone for context reasoning in ContReMAR. The structure of context reasoner is depicted in figure 2.

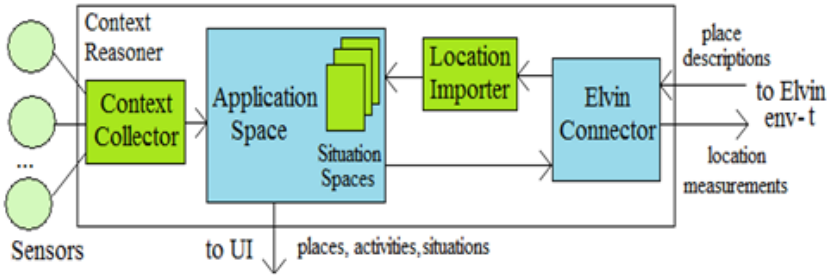


Fig. 2. Context Reasoner Architecture

In figure 2 the components specially designed for ContReMAR are depicted in green. Components reused from ECSTRA framework are depicted in blue.

Context reasoner consists of following components:

- **Context collector.** We redeveloped context collector component based on the similar component of ECSTRA framework. It is responsible for sensor fusion, proper reporting of

the newly arrived sensor data and preliminary analysis.

- **Application space.** Application space was reused from ECSTRA framework. Application space is responsible for handling the reasoning requests, which come from UI component. Application space contains complete current context information and encompasses situation spaces – the possible interpretations of sensor data.
- **Situation spaces.** Situations are generalizations of sensor information. Situation spaces are ESCTRA components, which are responsible for reasoning about one situation each. The situations of interest for this paper are the relevant places. However, in general case situations can also represent, for example, activities or events.
- **Location importer.** We developed location importer to ensure proper introduction of new places of interest (i.e. new situation spaces) into the application space.
- **Elvin connector.** Elvin connector is ECSTRA component (part of Elvin support in ECSTRA), designed to send and receive context information. It was extended comparing to ECSTRA in order to incorporate exporting and importing the place descriptions.

More details on ECSTRA and its components can be found in [BZ11a].

3.3 Location Analyzer

We designed and implemented location analyzer for extracting relevant places out of location data. Location analyzer is also responsible for relevance analysis, place type analysis and interacting with Google Places to detect the places in proximity of the location measurements cluster. Figure 3 shows location analyzer architecture. ContReMAR-specific components are depicted in green. The components, reused from ECSTRA, are depicted in blue. Location analyzer contains the following components:

- **Elvin connector.** The component originates in ECSTRA, but it was extended to handle new functionality of exchanging the situation descriptions. Elvin connector was already described in section 3.2.

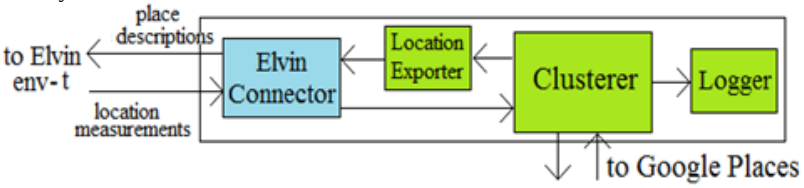


Fig.3. Location Analyzer Architecture.

- **Clusterer.** The clusterer component is one of the core components in ContReMAR. We designed the component to cluster location measurements in order to define the places of interest. Clustering was facilitated by the libraries of Weka toolkit [HF09]. Our application use density-based clustering, which was justified in section 2. The parameters of clustering algorithms are discussed in section 4. The clusterer is also responsible for analyzing the type of the place and communicating with Google Places service in order identify possible places, which the cluster can correspond to.
- **Location exporter.** Location exporter is responsible for translating clusters of points into situation descriptions and providing the descriptions to Elvin connector for further sharing with user device.
- **Logger.** We implemented loggesr component to provide detailed reports of how the application worked. It is mostly used for monitoring, debugging and evaluation purposes.

Next section discusses the advanced aspects of location analysis and provides the evaluation of ContReMAR application.

4 Evaluation

The proposed location awareness approach contains multiple parameters, like the minimum number of points for the cluster core, radius of the location point proximity, minimum time for the place to be relevant. In order to determine the best values of the parameters, extensive user studies are needed. Still unlabeled data, provided by Nokia MDC [LG12], allow performing some analysis and establishing some error bounds and parameter recommendations.

4.1 Experiments

We performed a series of experiments to prove and evaluate our approach. We used ContReMAR application with imitated user device as a testbed. During the experiment we simulated the mobile device and used Nokia MDC [LG12] data as an imitated sensor feed. The settings of every experiment, as well as the results, are reviewed in the subsequent subsections.

Experiment 1. Shall the location information be fused from multiple sources? Or is there any dominant source of location information? Location measurements come from two sources: GPS measurements and WLAN access at known spots. Figure 4 shows proportion of GPS location data in the total number of location measurements. The users to be depicted on the plot in figure 4 were chosen randomly (uniformly among all users in the Nokia MDC database [LG12]).

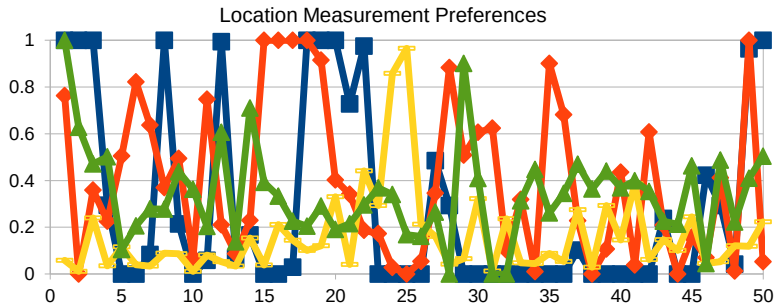


Fig 4. Proportion of GPS data in location measurements

The absence of obvious pattern in figure 4 allows deriving an important conclusion. Figure 4 proves that there is no dominant source of location information. Therefore, as it was expected, in practice we have to analyze both sources of location information and cannot simply use just one of those. This conclusion justifies the need for further experiments in order to determine, whether density-based clustering works well in presence of multiple-source location measurements.

Experiment 2. Can we use time thresholds to increase the precision of place recognition and minimize the number of false recognitions? What threshold value

should it be? The relevance of the place is influenced by the amount of time that the user spends at that place. If the user spends less than a minute at some place, most likely the user just passed it by. However, if the user spends around 15 minutes at some place, it is usually worth noticing. In the testbed application we implemented the following criterion – if the user spends less than certain amount of consecutive time at the location cluster, then the cluster is rejected and removed from consideration. The dependency between the number of detected clusters over time and the consecutive time threshold is presented in figure 5. The user was chosen randomly, but the trend holds for the vast majority of the users. We analyzed the possible time threshold values of 1 minute, 3 minutes, 5 minutes, 10 minutes and 15 minutes. The analysis of the trends in the number of recognized places allowed us adjusting the value of the time threshold.

We analyzed figure 5 according to the following criteria. The proper identification of relevant places should notice the most frequently visited places (like home, work or favorite lunch place) in first week or few weeks. After that the rate of detecting new places should slow down. If the rate stays high all the time, it can be a sign of numerous false recognitions. To summarize, if the number of recognized places grows with constant trend, it is a sign of possible massive false recognitions.

Figure 5 shows that the threshold of 1 minute leads to constant growth of recognized places. The threshold of 3 or 5 minutes also shows the same problem for most users. For the thresholds of 10 minutes of 15 minutes the number of recognized places almost stops growing in the first three weeks, and it matches the expected behavior of the correct location awareness algorithm. Therefore, the time threshold values of 10-15 minutes should be preferred.

Experiment 3. How can we configure clustering algorithm using unlabeled data? Can we do it by analyzing the fraction of revisited places? The efficiency of the place recognition algorithm can be evaluated by analyzing the number of revisited places. If the place is visited only once, it means that this place will just hamper recommendation and prediction algorithms. The possible unsupervised criterion for place recognition efficiency is the count of places, which were later revisited. Higher is that count, more significant places are detected.

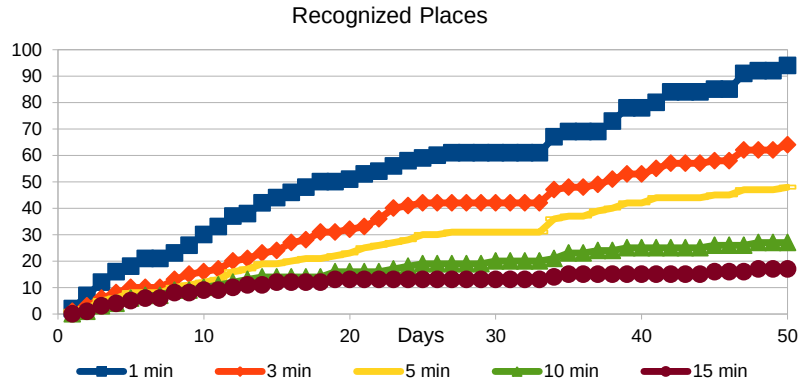


Fig. 5. Recognized places over time for random user, depending on the time threshold.

In table 1 we show the results of location measurements analysis for four randomly chosen users. We took all the places recognized in the first 25 days and analyzed how many times were they re-encountered during the first 50 days (i.e. during the 25 of active place recognition and 25 days after that, when we only analyzed the visits to the already recognized places). For the experiment we used the clustering algorithm with parameters *Eps*=50m, *MinPts*=3 and without consecutive time restrictions. Due to the very relaxed constraints the algorithm tends to recognize as a place everything that even remotely resembles a place of interest. We are going to refer to that algorithm as the benchmark clustering algorithm.

Table 1. Proportion of revisited places.

User (anonymized)	a	b	c	d
Revisited places (%)	49%	53%	38%	19%

As table 1 shows, the benchmark clustering algorithm will result in 50-60% of false recognitions (up to 80% in some cases). This information can be later used for auto-configuration purposes of location awareness algorithms. The parameters of the clustering algorithm can be preliminary evaluated by comparing it to the benchmark clustering approach. If the clustering approach under consideration leaves out in average 50-60% of clusters, recognized by the benchmark clustering algorithm, then it shows the expected behavior of a correct place clustering approach. Still, it should be noted that this criterion can be used only for preliminary estimation, and more precise parameter choice needs extensive user studies.

4.2 Demonstration and Evaluation Summary

To summarize, the experiments and demonstration lead to the following conclusions:

- GPS and WLAN location measurements are equally important for location awareness on the mobile phone. Density-based clustering is a feasible approach for detecting relevant places.
- In order to avoid false recognitions, the threshold can be put on the consecutive time that the user remained at some area. Using the threshold values of 1-5 minutes exhibits the signs of massive false recognitions, while the threshold values of 10-15 minutes show no visible problems.
- The clustering algorithm parameters can be evaluated by comparing its results to the benchmark clustering algorithm. The benchmark clustering algorithms is likely to produce from 50-60% to 80% of false recognitions. If the clustering algorithm differs from benchmark by that amount, it can be a preliminary indication of appropriate performance.

The space requirements do not allow extensive demonstration. For the proof of concept we show the following example. Figure 6 shows the results of location measurements analysis for randomly chosen user for day one. The system was able to determine most likely home and work places. For example, the application detected that the user works in EPFL Lausanne. The system suggested 3 possible places, where the user is likely to work (all of them are subdivisions of EPFL) and filtered out the places which couldn't be user's workplace (like the nearby streets).

5 Related Work

There is a large body of work in various techniques for trajectory mining, e.g. pattern discovery, similarity measures, clustering and classification for GPS data streams [HL08, JY08, LJ10].

Spaccapietra et. at [SP08] introduced "stops and moves" model for reasoning from GPS data. The application allowed detecting user visits to place of interest, but the user had to specify the relevant places manually. Palam et. al. [PB08] proposed spatio-temporal clustering method, based on speed in order to find interesting places automatically. SeMiTri system [YC11] focused on processing heterogeneous trajectories, integrating information from geographic objects and accommodating most existing geographic information sources. The GeoPKDD1 [AB07] address semantic behaviors of moving objects. Andrienko et. al. [AA11] developed a generic procedure for analyzing mobility data in order to study place-related patterns of events and movements.

An important novelty of our approach is fusion of information from multiple sources to build a comprehensive picture of user's location. Our approach also scales to large amounts of data to find the model of places of interest that allows feasible mobile real-time recognition using the measurements from multiple sources. One more novel feature of our approach is auto-labeling the places of interest based on both location and time analysis.

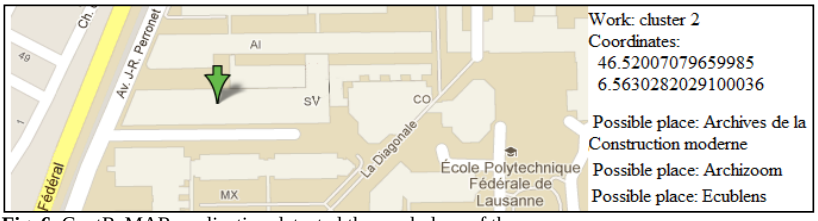


Fig. 6. ContReMAR application detected the workplace of the user

Next section concludes the paper and provides the direction of future work.

6 Conclusion and Future Work

In this paper we proposed a novel technique for mobile location awareness. In order to prove the feasibility and efficiency of the proposed approach we developed an application called ContReMAR. The outcomes of ContReMAR can enhance mobile activity recognition, context prediction and mobile context-driven recommender systems.

We used Nokia MDC dataset [LG12] as a simulated sensor feed in order to prove the soundness of our approaches, configure the proposed algorithms and evaluate the application performance. The evaluation showed the feasibility of density-based clustering approach to place identification and place recognition. Subsequent analysis led us to designing a set of experiments in order to test and evaluate the algorithms in absence of labeled data. In turn, it allowed us to configure the parameters of the algorithms.

We identified the following major directions of the future work:

- Improving the quality of place recognitions by performing user studies.
- Location-driven mobile activity recognition.

- Designing friendly user interface.

Present day mobile phones are able to sense the large amount of location information, as well as other diverse context. The proper analysis and verification of that information will significantly improve the capabilities of mobile devices and enable full-scale mobile location and context awareness, as well as support for advanced services and applications as has been demonstrated in [BZ12b].

Acknowledgements

We'd like to thank Shonali Krishnaswami for her support and mentorship. We'd also like to thank Basel Kikhia for providing valuable testing and visualization tools.

Chapter V

Structuring and Presenting Lifelogs Based on Location Data

Based on:

1. Kikhia, B., Boytsov, A., Hallberg, J., ul Hussain Sani, Z., Jonsson, H. and Synnes, K. Structuring and Presenting Lifelogs based on Location Data. Technical report. 2012. 19p. URL=http://pure.ltu.se/portal/files/40259696/KB12_StructuringPresentingLifelogs_TR.pdf, last accessed October, 30, 2012[67].

[6] Planned for submission to Personal and Ubiquitous Computing journal.

[7] My contribution to the report can be summarized as follows:

1. I proposed and prototyped place recognition approach.
2. Basel Kikhia and me together developed activity recognition method, analyzed evaluation results and calibrated the algorithms.

My input to the text of the article was second only to the first author, B. Kikhia. Together we wrote most of the text.

Foreword

This chapter addresses the research question 1 – how to find a mapping between context information and ongoing situations? Like chapter IV, this chapter addresses the approach of learning situations from unlabeled data. However, this chapter proposes an alternative approach comparing to chapter IV.

The solutions proposed in both chapters investigate learning situations out of unlabeled data and both solutions use density-based clustering for location inference. However, chapters IV and V take different approach to labeling.

Among other contributions chapter IV proposed a viable method for automated situation labeling in location awareness scenario. Manual labeling introduces intrusiveness, but can potentially provide more meaningful situation names. Labeling is not restricted to naming. For example, in the lifelogging scenario in this chapter locations and activities are first labeled using pictures from a SenseCam, and only then user gives meaningful names to them. For manual labeling the learned situations need to be presented to the user in an understandable manner. Moreover, intrusiveness should be kept to minimum. This chapter addresses lifelogging scenario and proposes an approach for learning important locations and activities out of unlabeled data. Also this chapter proposes an approach and an application, which ensure comfortable labeling and easy retrieval.

Structuring and Presenting Lifelogs based on Location Data

Abstract. Lifelogging techniques help individuals to log their life and retrieve important events, memories and experiences. Structuring lifelogs is a major challenge in lifelogging systems since the system should present the logs in a concise and meaningful way to the user. In this article the authors present a novel approach for structuring lifelogs as places and activities based on location data. The structured lifelogs are achieved using a combination of density-based clustering algorithms and convex hull construction to identify the places of interest. The periods of time where the user lingers at the same place are then identified as possible activities. In addition to structuring lifelogs the authors present an application in which images are associated to the structuring results and presented to the user for reviewing. The proposed approach allows automatic inference of information about significant places and activities, which generates structured image-annotated logs of everyday life.

Keywords: activity recognition, activity inference, lifelogging, clustering algorithms, SenseCam, GPS.

1 Introduction

Lifelogging is the act of digitally recording aspects and personal experiences of someone's life. Some people are interested in logging their life's activities for fun, medical purposes or diary applications [BL07]. It is important for many individuals to retrieve moments and events such as trips, weddings, concerts, etc. Reminiscing previous events among a group of people not only helps in remembering those events, but it also creates tighter social bonds and improves relationships among them [Do09]. Aiding memory is also one of the benefits that people gain by logging their life. For example, a lifelogging system can be used as an external memory aid that supports a person with memory problems by using reminiscence therapy [KH10a]. In reminiscence therapy the user reviews and talks about the day with someone, such as a caregiver. The review and discussion act both as a social activity and as assistance for the user to remember. For this purpose, and to improve events retrieval using lifelogging in general, lifelogs need to be properly structured. Structuring lifelogs is the primary issue being addressed in this article.

A natural way to structure lifelogs is in the form of activities; for example having lunch, sitting in the park, shopping, attending a seminar, etc. This structuring requires techniques for reasoning and inferring of activities from the logged data. The logged data is part of the lifelogs and the granularity, as well as the types of data, can vary. However, the basic context should be captured to infer activities. This basic context have been analysed and identified as identity, location, activity and time, where locations and activities are of special importance [KH10a][DA00]. Context data could be captured by mobile devices carried by the user such as wearable sensors. It is good, however, to use a single mobile device when logging as the number of devices the user needs to carry should be kept to a minimum.

Just structuring data into activities based on context may not be sufficient for efficient retrieval and to support people reviewing their life experiences. Both context (e.g. time, locations and places) and content (e.g. images) need to be aggregated and segmented into

the activities and be given semantic meaning. In previous work the authors have explored using known places to create this semantic meaning [KH10a]. However, this approach is limited to predefined places. A desired solution would be finding places of importance and then inferring activities automatically. In this article the authors introduce an approach to detect new places and then infer activities automatically relying solely on time-stamped location data. Location and time are rich and easily accessible sources of context information that are relevant to find places of importance, where the user spent significant time. Being for a period of time in a significant place might be an indication of some activities happened in the place. The first problem that the article addresses is: **"*How can places of importance be recognized and activities be inferred based on location data and time?*"**

Once lifelogs are segmented into activities, they can be annotated with content, such as images and descriptions. Images play a vital role in enriching the logs and in supporting reminiscence processes in a lifelogging system [FK02][Ch97]. Images can be captured automatically by purpose-built devices (e.g. SenseCam which is further described in Section 5) or by a smart-phone carried in a way that allows it to capture images. However, the information and the images still need to be presented to the user in a way that takes advantage of the structured lifelogs. The second problem that this article addresses is: ***"How can structured lifelogs be presented so the user can review and retrieve the life experiences?"***

The rest of this article presents the work done to address the problems listed in the introduction and is organized as follows: section 2 shows what algorithms have been used in this work to recognize new places. The calibration of the chosen place recognition algorithms is presented in section 3. Section 4 discusses the algorithm that has been used to infer activities. The development and deployment of the prototype application, which organizes the logs and presents them to the user, is the topic of section 5. Section 6 presents some of the related work and section 7 discusses the research questions. Finally, section 8 concludes the paper and presents the future work.

2 Recognizing places of importance

One of the problems addressed in this article is how to recognize places of importance for the purpose of structuring lifelogs. This is important because the places people visit contain hints towards the activities taking place. In fact, time-stamped location data can be used to recognize relevant places, and then infer activities based on identified places and time. In practice, areas where the user spends a significant amount of time can be seen as important locations or as activities done by the user at a specific location. One of the common approaches for discovering interesting patterns and data distributions from location data is density-based clustering algorithms [EK96][AB99]. For instance, these algorithms can infer information of areas where the user spent significant time when having location data logged by a mobile sensor carried by the user [PB08][AA02].

The proposed Place Recognition Algorithm relies on GPS points as a source of location data. The algorithm, however, is not restricted to GPS location information only. The adopted approach is depicted in Figure 1 and works as follows:

1. Raw time-stamped location data are used as an input for the approach.
2. Location data are clustered to identify places.
3. A convex hull is constructed over each cluster to estimate the geographical boundaries of the place.

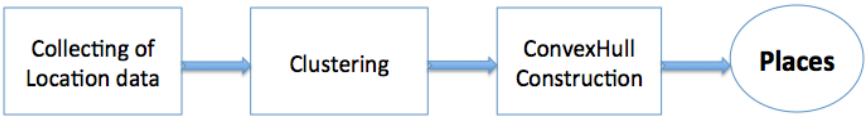


Fig. 1. New Places Recognition – Action Flow

The aim of the clustering algorithm is to identify places of importance to the user, which are previously unknown in the system. These places can be confirmed and labeled by the user while reviewing the lifelogs. If the user confirms a place, the system will add the coordinates that correspond to this place and define the place as a known one. The algorithm compares each GPS point with all previously known places. If the point belongs to a known place, the algorithm will remove it from the input set, but keep it for inferring activities later on. If the point does not belong to a known place, the algorithm will keep it in the input set for clustering. The GPS points in the input set are then clustered and aggregated regardless of time. Such clusters are signs of places where the user spent significant time. For example, the user might go to the office at different times of the day but the place is still the same.

The following requirements should be taken into consideration when choosing the clustering algorithm:

- The number of clusters is initially unknown. Different users have different numbers of visited places. The user can also visit a place that is never visited before. Algorithms which have pre-defined number of clusters will consequently not work in that case.
- Some GPS points might not correspond to a significant place for the user. For instance, the user might walk home from the working place. The location data, captured on the way home, do not belong to a significant place. The clustering algorithm should therefore allow a point not to be a part of any cluster.
- The algorithm should be noise-tolerant and resistant to outliers. Even though GPS points correspond to the user's position, outliers can appear when the user is inside a building when there are not enough satellites detected. These outliers should be identified as noise.

Density-based clustering algorithms satisfy all the aforementioned requirements. Two algorithms of density-based clustering approach were used by the authors: Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [EK96] and Ordering Points To Identify the Clustering Structure (OPTICS) [AB99]. OPTICS can be viewed as extension of DBSCAN. While DBSCAN provides faster runtime clustering, OPTICS supplies the developers with additional analysis tools and better visualization of the results [AB99]. Therefore, DBSCAN was implemented to cluster location data and then OPTICS-based analysis was used for algorithm configuration and parameters adjustment.

After the clusters are identified, the system constructs the convex hulls to estimate the geographical boundaries of the places. The convex hull of a set of points Q is the smallest convex polygon P for which each point of Q is either on the boundary of P or in its interior [CL09]. It is usually assumed that points in the set Q are unique and that Q contains at least three points that are not collinear. Rubber band analogy is often used for better understanding. Each point in Q is considered as being a nail sticking out from a board. The convex hull is then the shape formed by a tight rubber band that surrounds all the nails [CL09]. Figure 2 illustrates the view of the place clusters after implementing DBSCAN over the location data and constructing the convex hull.

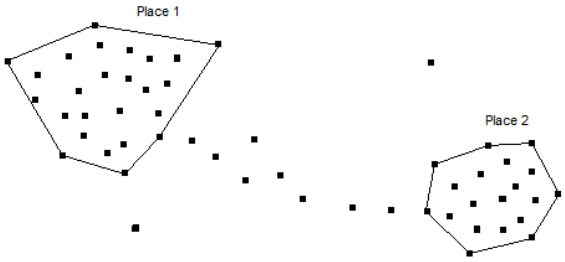


Fig. 2. Recognized places

3 Calibrating the Place Recognition Algorithm

Algorithms like the aforementioned density-based clustering algorithm have parameters that should be set in advance. Therefore, the parameters' values should be calibrated and tuned so the algorithm produces a minimum number of errors.

DBSCAN algorithm uses two parameters: the *Radius*, the range around a point where other points in that range are considered neighbours, and *MinPts*, minimum number of neighbours that a point needs in order to not be declared as noise. After setting the parameters, the algorithm forms clusters using the density of local neighbourhoods of points. This is done by selecting a point and then assigning all the points within its *Radius* (high destiny neighbours) to the same cluster. This approach is repeated for all the points resulting in many clusters with different arbitrary shapes. All points that do not belong to any cluster are considered noise.

A suitable set of parameters is the set that results in fewer numbers of place recognition errors. In order to evaluate the number of place recognition errors that correspond to different parameter sets, real-life data were collected and labeled manually, the possible error types were defined and then the performance of different parameter sets was estimated with respect to the identified error types.

3.1 Data Collection

A Windows Mobile application has been developed to log GPS tracks periodically every 30 seconds. If there are no satellites detected when logging, the application will do nothing and wait another 30 seconds to look for satellites. When connecting the logging device to a computer, the application transfers the logs as an XML file that contains longitude, latitude, logging time, speed, and number of satellites. The application then deletes the logs from the logging device so they do not interfere with new logs. Three users have done the data collection over a period of six months. The users were asked to carry a mobile device, with the application installed, during the day. By the end of the day, the user connects the mobile to a computer to transfer the logs.

Relying on time-stamped GPS data has some limitations. Firstly, GPS data might be noisy and not accurate if few satellites were detected when logging the location. To overcome this limitation, the logged GPS data with less than 4 satellites are ignored. Having 4 satellites or more showed good results when doing manual analysis. Blocking the view of the GPS receiver by another object is another limitation. This happens sometimes when the user has keys, for instance, together with the mobile device in the same pocket.

This case results in missing some location data or having noisy data. During the data collection period, the users were advised to not have any other object next to the mobile device to reduce the risk of blocking the access to satellites.

The users were also involved in the evaluation process to ensure that the algorithm's results correspond to significant places.

3.2 Error Types

Manual analysis of the collected data revealed that there are 4 types of possible errors:

1. The algorithm detected a cluster that does not correspond to any real-life place. The impact of this error is the least severe of all four. It is possible for the user to manually fix this by discarding this cluster when reviewing the lifelogs. This kind of error can also result in detecting non-existent activities on later activity inference steps, but this can be also fixed when reviewing the logs. However, this problem can still result in distraction and waste time for the user. The amount of errors of this kind tends to grow with increasing the *Radius* or with decreasing *MinPts*.
2. The algorithm merged two places into one. This problem causes distraction to the user and it can disrupt the activity inference process. The number of errors of this kind tends to grow with increasing the *Radius* or with decreasing *MinPts*.
3. The algorithm separates one place into two different ones. This problem is opposite to the previous one. The number of errors of this kind tends to grow with decreasing the *Radius* or with increasing *MinPts*.
4. The algorithm did not detect an essential place. This error type is the most serious one because the activities in that place will be lost as well. The number of errors of this kind tends to grow with decreasing the *Radius* or with increasing *MinPts*.

3.3 Parameter Values

25 randomly chosen logs were analysed to determine the best parameter values of the DBSCAN algorithm. Each log has data collected during one day. Logs were manually analysed and essential places were identified based on observation.

The DBSCAN algorithm has been implemented using JavaScript and the results have been shown through a web application and manually processed to identify errors of different types. The application shows a map with all collected points during the day on the left side, and the clustering results after applying DBSCAN based on the *Radius* and *MinPts* on the right side. Figure 3 presents part of the results when running DBSCAN on one selected log with 20 meters as a *Radius* and 3 points as *MinPts*. The points that are marked by 1 belong to one cluster while the points that are marked by 2 belong to another cluster.

Different reasonable values of the *Radius* and *MinPts* were tested to find out what errors they produce. For each log, the following parameters sets were considered: every possible *MinPts* from 2 to 20 with the step of 1, combined with every possible *Radius* from 5 meters to 30 meters with the step of 5. The aforementioned trends of errors (error types in section 3.2) allowed finding the best values without checking all the combinations of every *MinPts* with every *Radius*. Those heuristics significantly reduce the number of parameter combinations for manual testing, and practically allow to check just ~10-30 parameter combinations per log. The aim is to find the minimum values of *MinPts* and the *Radius* that result in fewer numbers of errors for each log. The priority is to reduce the errors of type 4 when the algorithm does not detect essential places. After determining the best values for each log, the average of those values is calculated to find out a representative value for the

whole set of logs. Table 1 shows the best combination for minimum values of the *Radius* and *MinPts* for each log followed by the average values.

The average value for *MinPts* is 3.24, while it is 12.8 for the *Radius*. Since increasing *MinPts* might result in increasing the number of undetected places, the value 3.24 is rounded to 3. Thus the parameter values that yield the best results for the DBSCAN algorithm are: 3 for *MinPts* and 12.8 meters for the *Radius*.

As an additional method of parameters adjustment the OPTICS algorithm was used [AB99]. OPTICS operates with the same parameters as DBSCAN, but while *MinPts* should be specified manually, the neighbourhood radius can be adjusted by the analysis of reachability plots. The plots visualize the structure of clusters and sub-clusters. Analysis of OPTICS reachability plots can confirm the values of the DBSCAN parameters. The analysis also gives an indication of how stable the clustering approach is. To perform OPTICS-based analysis, the WEKA toolkit was used [HF09]. WEKA is a machine learning software that has a collection of visualization tools and algorithms for data analysis and predictive modelling.

Table 1. Summarization of the logs analyses

GPS Log#	MinPts	Radius (meters)
1	2	10
2	4	10
3	2	15
4	3	10
5	3	5
6	2	10
7	3	30
8	4	5
9	3	5
10	5	10
11	2	5
12	3	30
13	2	20
14	2	5
15	4	15
16	3	10
17	3	5
18	2	15
19	3	10
20	6	20
21	4	5
22	3	15
23	2	10
24	5	25
25	6	20
Average	3.24	12.8

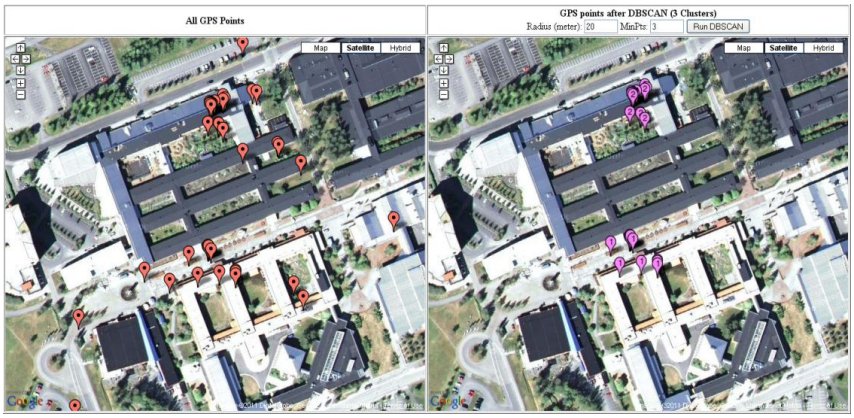
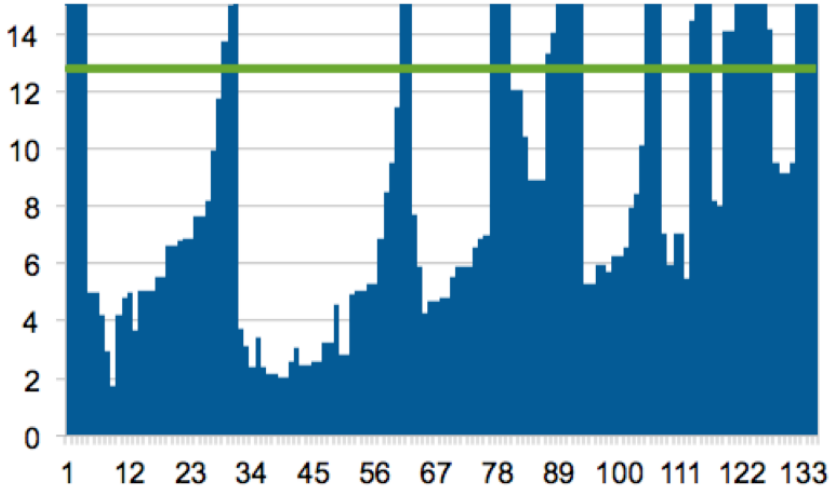
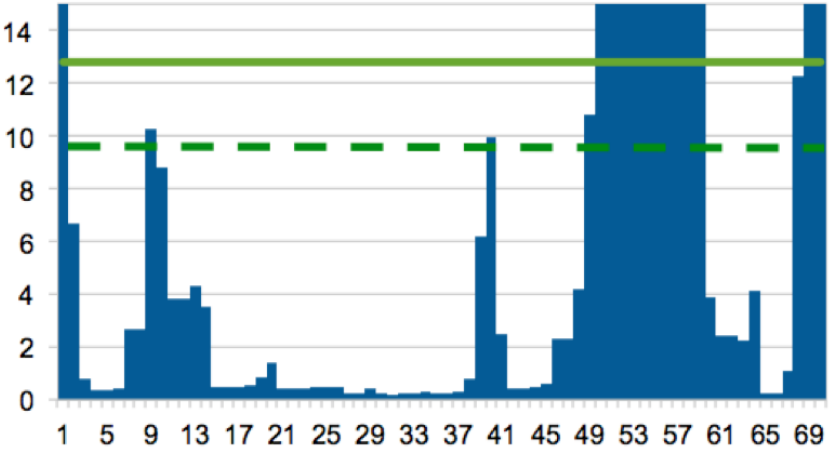


Fig. 3. DBSCAN implemented in a web application

Two sample reachability plots are depicted in figure 4. Reachability plots (a) and (b) correspond to two different GPS logs chosen out of the 25 logs of testing. *MinPts* is set to 3 and the green threshold lines correspond to the *Radius* value of 12.8 meters. The plot areas below the threshold line correspond to clusters. In the reachability plot (a) there are 8 clusters, while there are 2 clusters in the reachability plot (b). This visualization can identify how significant change of the neighbourhood radius is required to obtain different clustering results. For example, the leftmost cluster of the figure 4 (b) would have been recognized as 3 separate clusters if the neighbourhood radius is set below 10 meters. If the reachability plots contain many values near the threshold, it means that the clustering is unstable, and the number and the structure of the clusters can be heavily influenced by random fluctuations in GPS measurements. The analysis of reachability plots allowed the authors to conclude that for the vast majority of the testing samples it takes significant change of the neighbourhood radius to alter the clustering results. Therefore, the chosen parameter values provide stable clustering and low sensitivity to random factors.



Reachability Plot (a). The green line represents the threshold value of the Radius *12.8 meters.* 8 clusters are recognized (the white areas below the threshold)



Reachability Plot (b). The green dashed line represents the threshold value of the Radius if it is changed from *12.8 meters* to less than *10 meters.* The leftmost cluster will be recognized as 3 clusters.

Fig. 4. Reachability plot visualization when using OPTICS

4 Inferring Activities

Once places of importance have been identified then they can be used to infer activities. To reason about activities, the following properties of an activity are identified:

1. An activity occurs at a place. The place can be a new one, which was just obtained by the place recognition algorithm, or a previously known one for which the geographical boundaries are known. This property leads to several important consequences:

- The GPS points that correspond to the same activity are defined as a subset of the GPS points that correspond to the same place. This reduces the scope of the algorithm from the entire set of GPS points to just points within places.
- If the user leaves a place, even for a short time, the activity is interrupted. This consequence can help in distinguishing several activities that happened in the same place at different times.

2. An activity takes a certain amount of time.

Based on these properties, a set of GPS points is an indication of an activity if:

- The points belong to the same place.
- The points are sequential in time.

The main idea of the activity inference algorithm is to decompose all place clusters into sub-clusters that do not overlap with each other in time. Place cluster refers to the set of GPS points within the place. Overlapping occurs when the user leaves place *A*, for example, to place *B* then comes back later to place *A*. For instance, the user spends time in the office from 8 AM to 1 PM, goes to a restaurant for lunch from 1 PM to 2 PM, back to the office from 2 PM to 5 PM and finally goes to the same restaurant from 5 PM to 7 PM for dinner. The timeframe of the GPS points captured when being in the office will be from 8 AM to 5 PM while the ones captured when being in the restaurant will be from 1 PM to 7 PM. Decomposing the office cluster and the restaurant cluster into sub-clusters that do not overlap in time will result in 4 sub-clusters that represent 4 different activities:

- From 8 AM to 1 PM (being in the office)
- From 1 PM to 2 PM (being in the restaurant)
- From 2 PM to 5 PM (being in the office)
- From 5 PM to 7 PM (being in the restaurant)

To identify activities that happened at certain places based on the definitions in this section, the following algorithm has been introduced. For any place *A*, the earliest captured GPS point at this place is marked as *A.begins*, and the latest captured GPS point at this place is marked as *A.ends*. The fact that GPS points are naturally ordered by time makes calculating the timeframe easy. Clusters that correspond to the visited places are added to the input set of the activity inference algorithm.

Algorithm:

1. *Take the first element of the input set (and remove it from the set). Let it be cluster A.*

2. *Search through the entire remaining parts of the input set. Search goes on until we find another element whose timeframe overlaps with the timeframe of A. Let it be cluster B.*

3. *If no overlapping cluster can be found:*

3.1. Add cluster A to the result set.

3.2. If the input set is not empty, go to step 1.

3.3. If the input set is empty, go to step 5.

4. *If an overlapping cluster can be found:*

4.1. Remove cluster B from the input set.

4.2. Order the following points in time in ascending order: A.begins, B.begins, A.ends, B.ends. Let's refer to the ordered time variables as time1 ≤ time2 ≤ time3 ≤ time4.

4.3. Add to the input set the following sub-clusters of cluster A:

1) Sub-cluster within timeframe [time1, time2]

2) Sub-cluster within timeframe [time2, time3]

3) Sub-cluster within timeframe [time3, time4]

4.4. Add to the input set the following sub-clusters of cluster B:

1) Sub-cluster within timeframe [time1, time2]

2) Sub-cluster within timeframe [time2, time3]

3) Sub-cluster within timeframe [time3, time4]

4.5. If the input set is not empty, go to step 1.

5. *End the algorithm and return the result set.*

Output: a set of sub-clusters that do not overlap in time, each sub-cluster represents an activity.

The final timeframe of an activity can be obtained in a straightforward manner after the GPS points are attributed to activities; the time between the earliest point and the latest point of the activity. If a sub-cluster contains only one GPS point (lets refer to this only point as *P*), then the timeframe for this activity is counted as follows:

- If the sub-cluster is the earliest one, the timeframe of the activity is counted as the time between *P* and the next collected point after *P*.
- If the sub-cluster is the latest one, the timeframe of the activity is counted as the time between *P* and the previous collected point before *P*.
- If the sub-cluster is not the earliest or the latest one, the timeframe of the activity is counted as the time between the previous collected point before *P* and the next collected point after *P*.

Figure 5 illustrates the view of activities after decomposing Place 1 (which is shown in figure 2) to sub-clusters that represent activities.

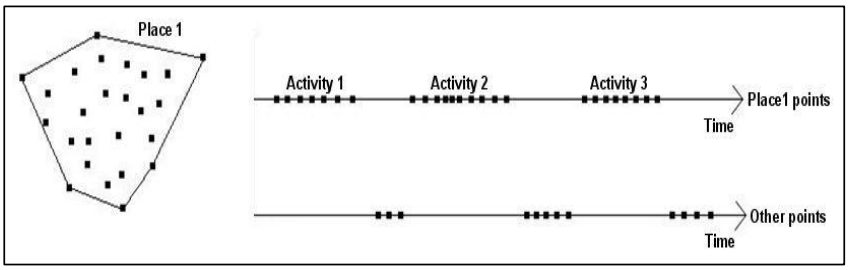


Fig. 5. Recognized activities within a place

5 Implementation and Deployment

For lifelogs to be useful they need to be structured and presented to the user in a way that will give a good overview of content and data. The proposed solution for structuring lifelogs is to identify places of importance, infer activities, and then associate images with the places and the activities. A prototype application to demonstrate how this could be done, used with six months' worth of captured lifelog data, is therefore presented in this section.

The prototype consists of mobile devices for capturing images and contexts, and an application for reviewing the gathered data. The mobile device for capturing images that is being used in this prototype is called SenseCam, which is depicted in Figure 6.

SenseCam is a wearable digital camera, which keeps a digital record of the activities that the person experiences [KB10][GW04][BD08][DC08]. All recordings are automatically logged without the user's intervention and therefore without any conscious effort [GW04]. SenseCam contains a number of different electronic sensors which can be used to collect data for the lifelogs: light-intensity and light-color sensors, a passive infrared (body heat) detector, a temperature sensor, and a multiple-axis accelerometer. Certain changes in sensor readings can also be used to automatically trigger a photograph to be taken [HW06] which helps capturing things of significant importance. SenseCam does not feature a GPS sensor, so location data was instead captured by a smartphone, synchronized with the SenseCam.

When connecting the two devices, the SenseCam and the smartphone, to a computer with the prototype application installed, the system performs the following steps:

Fig. 6. SenseCam worn around the neck

1. Transferring the logs in the form of XML. The logs consist of time-stamped GPS data and time-stamped images.
2. Analyzing the GPS data to identify periods of time where the user visited known places during the day.
3. GPS points that do not correspond to any of the known places are aggregated, using the DBSCAN algorithm, into clusters that represent new places. The *Radius* is set to 12.8 meters and *MinPts* is set to 3.
4. Inferring activities based on the places using the method presented in section 4.
5. Associating SenseCam images with the recognized places and the inferred activities based on time.
6. Showing the results on the main interface in a chronological order.

The prototype application associates SenseCam images with auto-recognized places and inferred activities. The middle image of each group of images that belongs to a place or to an activity is chosen as a representative image. When reviewing, the application shows not only the images, associated with places and activities as content, but also the location context data. This can improve the reviewing process as it combines context with content data. Figure 7 shows the main interface of the application after transferring the logs of one day. This interface consists of 2 columns, where one column presents places and the other one presents activities.

5.1 Reviewing Places

Figure 8 shows the place page when reviewing. When reviewing a place, the system shows the constructed convex hull from the GPS points that correspond to the place. In addition, SenseCam images that have been captured when the user was in the place are shown.

The user can choose a representative image for the place using the available SenseCam images. The user can also upload own images from an external source. If the user confirms the place, the system will save the chosen image as the representative one together with the coordinates that correspond to this place. Thus the place will be known and detected automatically by the system if the user visits it again. As the user continues to review the data, more places will be added to the list of known places of importance. This will improve the system's knowledge of important places, which will increase the level of automation in detecting the user's movements.

5.2 Reviewing Activities

When reviewing an activity, the system presents all SenseCam images that have been captured during that activity. The system also shows all the GPS points that correspond in time to this activity on a map. It is possible for the user to choose certain images to associate with the activity among the whole set of images. Figure 9 shows the activity page when reviewing.

Reviewing activities differs from reviewing places in the following aspects:

- It is possible to annotate an activity with many images, while a place can be annotated only with 1 image.
- The coordinates that correspond to an activity will not be saved in the system if the user chooses to save the activity.
- All the activities that happened in a place will be associated with this place for later context-dependent retrieval.

Saved activities can be retrieved later based on date/time or places. This means that the user can recall all the activities that happened at a certain time or in a certain place.

6 Related work

In this article the authors have shown how to recognize places of importance, how to use them to structure lifelogs in the form of activities, and how to present lifelogs to the user in a structured way. This section describes some of the work that has been done by others, which is related to the work presented in this article.

The DBSCAN algorithm, together with OPTICS, was used by the authors when clustering the location data. Another common clustering approach is K-means. K-means is a method of cluster analysis that aims to partition n observations into k clusters [WX07].

Fig. 7. The main interface of the lifelogging application

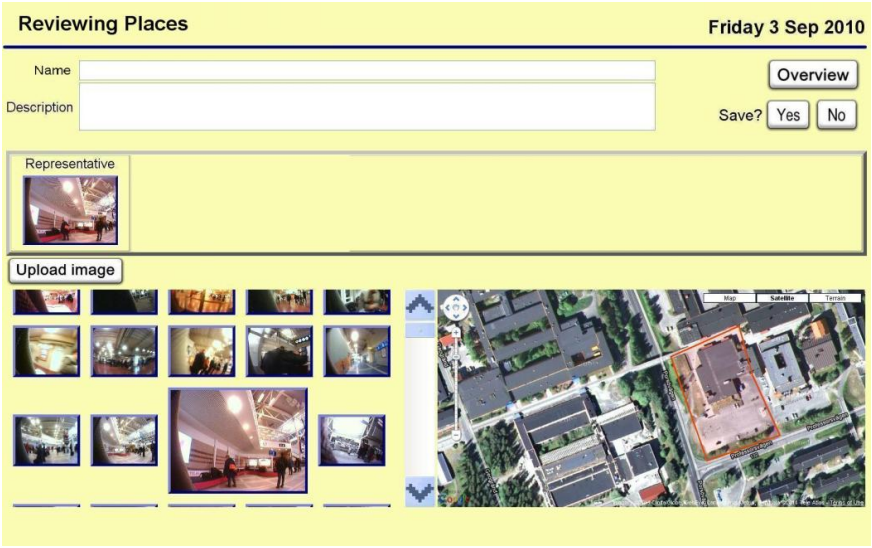


Fig. 8. Reviewing a place within the lifelogging application

Fig. 9. Reviewing an activity within the lifelogging application

This method assigns *k* initial means randomly, and then it goes over all the observations and assigns them with the nearest mean, which results in having *k* clusters. The mean value of each cluster is calculated, and then the method is repeated until the error term is deemed small or the decrease is below a threshold. This method is very sensitive to noise, and the number of the clusters must be known in advance. K-means also rely on the random initialization of the means that makes it non-deterministic [ZF04]. Ashbrook et al. [AS02] used a variant of the k–means clustering algorithm that used GPS data in defining locations of the user. All the points within a pre-defined *Radius* are marked, and the mean of these points is computed. The system will do calculation again based on the mean and the given *Radius* to find a new mean value. When the mean value is not changing any more then all points within its *Radius* are placed in a cluster and removed from consideration. This approach is repeated until there are no more GPS points left. The main limitation of this approach is that the *Radius* should be set in advance and then the clustering algorithm will rely on that value. Density-based algorithms overcome the limitations of the K-means clustering method [EK96]. The advantages of using DBSCAN over K-means are mentioned by Zhou et al. [ZF04]: DBSCAN is less sensitive to noise, it allows clusters of arbitrary shape, and it provides deterministic results.

Alvares et al. [AB07] presented an approach to add semantic information to trajectories. Trajectories are decomposed into stops and moves where a stop is a set of points that is transformed into a geographic object with a meaning, and a move is a small part of a trajectory between two stops. Candidate stops are identified in advance and the output of the work was a semantic trajectory dataset. Palma et al. [PB08] extended the work presented in the paper [AB07]. Palma et al. used a variation of the DBSCAN algorithm to discover interesting places in trajectories, which are previously unknown. Trajectories are observed as a set of stops and moves, where stops are more important. The authors in [PB08] calculated the distance between points along the trajectory instead of using Euclidean distance, and they used minimal time instead of using minimal number of points

MinPts, for a region to be dense. The absolute distance (*Eps*) is used to calculate the neighbourhood of a point. The choice of Eps requires knowing the percentage of location points, which corresponds to stops. In our work the choice of *Eps* is based on analysis and observation of real-life data. In contrast to [AB07] and [PB08], our work considers activity inference and contains the methods to visualize and label the found places and activities.

Andrienko et al. [AA11] defined the trajectory of an object as temporally ordered position records of that object. The authors in [AA11] looked at the characteristics of the movement such as instant speed, direction, travelled distance, displacement and the temporal distances to the starts and ends of the trajectories. These characteristics are then represented as dynamic attributes that can give an indication of movement events. For instance, having low speed in some areas can be an indication of specific events belonging to those areas. The events are clustered according to their positions and time, and then used to extract places. Repeated occurrences of events in places are calculated by means of density-based clustering, and those places are defined as interesting ones to the user. The result was defining places of interests from mobility data by analysing place-related patterns of events and movements. However, the work presented in [AA11] relied on the data collected by many users in the area, while our work is designed for detecting and logging personal preferences, so activities in our work represent the personal life experiences of the user. In addition, no prototype application was done in [AA11] so the user cannot review and save the detected places and events for later retrieval.

Another work was presented by Wolf et al. [WG01] that relied on the loss of the GPS data as indication of buildings. Losing GPS signals within a given *Radius* on three different occasions is interpreted by the system as a building has been entered. This work identified only buildings and provided no detection for outdoor places or any activities.

The effect of using locations and images on memory recall has been tested by Kalnikaite et al. [KS10]. In their work, SenseCam images are associated with locations based on time and then presented to the user through an application. However, images are associated without the use of any particular clustering techniques. Thus if a SenseCam timestamp falls within 50 seconds of a GPS timestamp, that image and the GPS point will be paired together. Another application that presents groups of images on a map based on their locations has been created by Toyama et al. [TL03]. All the images are tagged by the location data and stored in a database, and then the application groups the images and shows them on the map based on the tagged location. Locations can be cities, streets, or user-defined places. This application lacks the automatic detection of important places as it relies mostly on the tagged data of the images.

In summary, the work presented in this paper extends the previously mentioned works by presenting techniques that cluster location data and then grouping a sequence of images based on the clustering results. The authors combine recognition of places with inference of activities relying solely on time-stamped location data. Context and Content data are also combined and visualized through a prototype application, so the user can mark places and activities that happened at interesting stops to retrieve them later on. In addition, having a stream of images can help the user, when reviewing through the proposed application, in naming what activity happened in the place.

7 Discussion

This section discusses the results of the efforts with respect to the research questions.

The first addressed question is: **"*How can places of importance be recognized and activities be inferred based on location data and time?*"**

Places can be recognized relying solely on time-stamped location data using the DBSCAN algorithm. DBSCAN aggregates GPS points into clusters based on the density of points. The authors calibrated the density-based algorithm based on data collected by three users over a period of six months. The best parameter values for DBSCAN that result in fewer numbers of place recognition errors are 12.8 meters for the *Radius* and 3 points for *MinPts*. OPTICS algorithm is also used to ensure that the chosen parameters values for DBSCAN provide stable clustering results. The DBSCAN algorithm results in clusters that represent places visited by the user. After the clusters are identified, the system constructs the convex hull to estimate the geographical boundaries of the recognized places.

Activities are inferred based on the known places and the essential places that are defined in the previous step. An activity is represented by a set of GPS points which belong to the same place and which are sequential in time. The system searches within the defined place clusters and splits them into sub-clusters that do not overlap in time. Each sub-cluster represents an activity that happened in a certain place at a certain time. The timeframe of each activity is the time between the earliest point and the latest one within the sub-cluster. A cluster, which represents a place, might be divided into several sub-clusters, which represent activities happened in the same place at different time.

The second addressed question is: ***"How can structured lifelogs be presented so the user can review and retrieve the life experiences?"***

The lifelogs, which are structured based on places and activities, are presented through a prototype application that answers the following questions:

- When did the activity take place? The timeframe of the activity is presented based on the identified corresponding time-stamped GPS points.
- Where was the activity? The place where the activity happened is presented on the map based on a convex hull of the corresponding GPS points.
- What was the user doing? The presentation of the activity is based on the auto-captured images, which were taken at the time of the activity.

A SenseCam can be used to capture images automatically while a mobile device can collect GPS points during the day of the person. The system transfers all the logs when connecting those portable devices to a computer, and then defines places and activities based on the GPS data. SenseCam images are then associated with those places and activities based on time and presented to the user for reviewing and adjustment. Adding SenseCam images, as content, to the clustering results helps the user in naming places and activities when reviewing.

If the user confirms a cluster as a place, the coordinates that correspond to this place are saved and the place will be known and detected automatically next visit. Therefore the system can improve its knowledge about the user's preferable places. Saving activities will just save the data and make it available for later retrieval. The system thus presents the structured lifelogs as places and activities associated with SenseCam images. The system helps the user to retrieve or share previous moments in life based on places or time. For example, the user can review all the activities that happened in a certain place, such as the university, or at a certain time, such as the New Year eve.

8 Conclusion and Future Work

This article presented a novel approach that relies on location data and images to organize the lifelogs of someone's life. Location data provides a context source that can be used to

recognize places and infer activities. Images, as content data, can be then associated with those recognized places and inferred activities, and be presented to the user for reviewing and adjustment. The introduced prototype system structures and presents lifelogs based on places, activities and images that can be available for later retrieval. The system therefore provides a digital tool for people to reminisce and share their life.

The next stage of our work is improving the inference of activities within the lifelogging system using the same set of devices. Sensor-readings in SenseCam can be used with image processing techniques to better reason about daily activities. This will also help the system distinguishing between different activities that usually happen in the same place, which will improve the activity inference task.

Chapter VI

Formal Verification of Context and Situation Models in Pervasive Computing

Based on:

1. Boytsov, A. and Zaslavsky, A. Formal verification of context and situation models in pervasive computing. *Pervasive and Mobile Computing*, Volume 9, Issue 1, February 2013, Pages 98-117, ISSN 1574-1192, 10.1016/j.pmcj.2012.03.001.
URL=http://www.sciencedirect.com/science/article/pii/S1574119212000417, last accessed May, 08, 2013.

2. Boytsov, A. and Zaslavsky, A. Formal Verification of the Context Model – Enhanced Context Spaces Theory Approach. Scientific report, 2011, 41 p.
URL=http://pure.ltu.se/portal/files/32810947/BoytsovZaslavsky_Verification_TechReport.pdf, last accessed October, 30, 2012.

Foreword

Previous chapters addressed different ways of defining situations and answered the first research question: how to derive a mapping between context information and ongoing situations? This **c**hapter addresses the second research question: once the situations are identified, how to prove, that the derived mapping is correct? This chapter answers the research question 2 by proposing, proving and evaluating a novel concept – verification of situation models. Verification allows formally proving that a situation definition does not have error of certain kind, or (if there is an error) derive a counterexample – particular context features that will cause situation awareness inconsistency.

Formal Verification of Context and Situation Models in Pervasive Computing[8]

Abstract. Pervasive computing is a paradigm that focuses on availability of computer resources anytime anywhere for any application and supports non-intrusive integration of computing services into everyday life. Context awareness is the core feature of pervasive computing. High-level context awareness can be enhanced by situation awareness that represents the ability to detect and reason about the real-life situations. In this article we propose, analyze and validate the formal verification method for situation definitions and demonstrate its feasibility and efficiency. Situations are often defined manually by domain experts and are, therefore, susceptible to definition inconsistencies and possible errors, which in turn can cause situation reasoning problems. The proposed method takes as an input properties of situations and dependencies among them as well as situation definitions in terms of low-level context features, and then either formally proves that the definitions do comply with the expected properties, or provides a complete set of counterexamples – context parameters that prove situation inconsistency. Evaluation and complexity analysis of the proposed approach are also presented and discussed. Examples and evaluation results demonstrate that the proposed approach can be used to verify real-life situation definitions, and detect non-obvious errors in situation specifications.

Keywords: context awareness; situation awareness; context spaces theory; situation algebra; verification.

1 Introduction

Pervasive computing paradigm aims to integrate computing services gracefully into everyday life, and make them available everywhere and at any time. Partial implementations of this approach are, for example, ambient intelligence systems (like smart homes or smart offices), PDAs, social networks. One of the foundational features of pervasive computing is context awareness. Context awareness can be further enhanced by the concept of situation awareness – generalization of context information into real-life situations.

Manually-defined specifications of situations are usually clear to understand and easy to reason about. However, the creation of situation specifications is a resource and effort consuming work. One of the main problems of manual definition is the possibility to introduce a situation specification error. The specification errors can result in inadequate situation reasoning and internal contradictions in context reasoning results. The theoretical solution and practical implementation presented in this article allow formally verifying specifications of situations using the expected situation relationships, and, if the verification detected an error, allow deriving counterexamples – the exact context properties that will lead to inadequate situation awareness results. The relationships that are being verified may include *contradiction* (e.g. verify that the situation "Driving" cannot co-occur with the

[8] This part contains merged publications [BZ11c] and [BZ12b]. The article [BZ12b] was taken as a baseline, but the proofs of lemmas and algorithms were taken from the report [BZ11c]. References and formulas were renumbered accordingly.

situation "Walking"), *generalization* (e.g. verify that by specification the situation "InTheLivingRoom" implies the situation "AtHome"), *composition* (e.g. verify that by specification the situation "InTheCar" consist of "OnBackSeat" and "OnFrontSeat"). The detailed description of possible relationships that can be verified is formulated in section 3.1 and the motivating example is provided in section 3.2.

To the best of our knowledge this research is a pioneering approach for verification of integrity, consistency, non-contradiction and adequacy of a context model, which represents the understanding of internal and external environment of a pervasive system. The proposed capability to verify the context model detects and eliminates errors during pervasive computing system development and at runtime, and therefore results in more reliable and dependable systems. This approach is applicable to context models based on context space theory and in general to broader class of context models, e.g. decision trees.

This article is structured as follows. Section 2 addresses the basics of context spaces theory, the background theory of this article, and introduces some additional definitions that will be used throughout the article. Section 3 introduces the challenge of formal situation verification and proposes the general approach to address that challenge. Section 4 proposes and analyzes improved situation representations using context spaces theory, which later will be used as a basis for the verification algorithms. Section 5 proposes the verification approach. Section 6 contains theoretical complexity analysis and practical evaluation of the proposed verification approach. Section 7 discusses related work and distinctive and unique features of the proposed verification approach. Section 8 contains summary, future work directions and concludes the article.

2 The Theory of Context Spaces

2.1 Basic Concepts

The context spaces theory (CST) was introduced by Padovitz et. al. [PL08a][PL08b]. CST uses spatial metaphors to represent context as a multidimensional space and to achieve insightful and clear situation awareness. In this section we summarize the definitions and concepts of context space approach. These concepts were redeveloped and enhanced to provide more solid basis for the verification techniques, which are the focus of this article.

A *context attribute* [PL08a] is a feature of context and context can be represented by multiple context attributes. Context attributes correspond to respective domains of values of interest. For example, in smart home environment information like air temperature, energy consumption and light level can be taken as context attributes. Context attributes can be numeric (e.g. noise level, air humidity, illuminance level, water temperature), non-numeric (e.g. on/off switch position, door open/closed) and mixed (that can have both numeric and non-numeric values at different time, e.g. air conditioner setup – particular temperature or the value "Off"). In case if a context attribute is missing (for example, due to unavailability of the sensor), its value is usually set to *undefined*, which is treated just as a possible non-numeric value.

A context attribute can be metaphorically represented as an axis in a multidimensional space. In this article a certain value of context attribute, taken with respect to uncertainty, is referred to as *context attribute value*. In the simplest case context attribute values are particular points on the context attribute axis – single values, numeric or non-numeric, without any attached estimations of uncertainty.

Situation reasoning requires testing, whether the context attribute value is within some

interval. Generalized concept of interval over some context attribute *ca* can be defined in one of the following ways:

1. If *ca* is numeric or mixed, the interval is just a numeric interval. The borders can be included or excluded arbitrary. The possible formats are: [*a*;*b*], (*a*;*b*), [*a*;*b*) or (*a*;*b*], where $a,b \in \mathbb{R}$, and $a \leq b$.
2. If *ca* is non-numeric or mixed, there are 2 possible formats for a generalized interval.

2a. The generalized interval contains a set of possible values: $\{a_1, a_2, \ldots, a_N\}$, where a_i are non-numerical context attribute values. If the context attribute has one of those values, it falls within the interval. It should be specifically noted, that checking for undefined context attribute also falls under that category.

2b. The generalized interval contains a set of prohibited values: $\neg\{a_1, a_2, \ldots, a_N\}$, where a_i are non-numerical context attribute values. If the context attribute does not match any of the values $a_1, a_2, \ldots, a_N$, it falls within the interval.

From now and on, when referring to any interval over context attribute axis, the generalized concept of interval will be implied.

Two generalized intervals overlap, if there exists a context attribute value that belongs to both intervals.

A multidimensional space, which comprises multiple context attributes as its axes, is referred to as *application space* or c*ontext space* [PL08a].

The entire vector of relevant context attribute values at a certain time is referred to as *context state* [PL08a]. In spatial representation, the context state can be viewed as a point in multidimensional context space.

The concept of *situation space* is developed in order to generalize context information and provide higher-level reasoning. Situation spaces are designed to formalize real-life situations and allow reasoning upon real-life situation using the sensory data. A situation space in original CST situation definition can be identified as follows [PL08a]:

$$S(X) = \sum_{i=1}^{N} w_i * contr_{S,i}(x_i) \quad (1)$$

In formula (1) *S(X)* is a confidence level of situation S at a certain state X. The context state X includes context attribute values x_i that are relevant for situation S. The coefficient w_i represents the weight of i-th context attribute contribution to the total confidence of the situation S. The number of relevant context attributes is N, and $contr_{S,i}(x_i)$ is a function that measures the contribution of i-th context attribute to the total confidence of the situation S.

Usually the contribution function resembles a step function over a certain context attribute. Formula (2) shows the contribution function format.

$$contr_{S,i}(x) = \begin{cases} a_1, x \in I_{i,1} \\ a_2, x \in I_{i,2} \\ \ldots \\ a_m, x \in I_{i,m} \end{cases} \quad (2)$$

In formula (2) $I_{i,j}$ are various generalized intervals over i-th context attribute (possibly including the test for missing context attribute value). Intervals over the same context attribute should not overlap. Also the intervals $I_{i,1} \ldots I_{i,m}$ should cover entire i-th context attribute, i.e. any possible context attribute value *x* should belong to an interval from $I_{i,1} \ldots I_{i,m}$ set. Contribution levels a_i are usually within [0;1] range. A contribution level a_i

can as well be set to *UD* (undefined) for some intervals. Usually undefined contributions correspond to missing context states. The presence of any undefined contribution makes the entire situation confidence value undefined.

In order to achieve Boolean situation reasoning results, a threshold is often applied on top of the confidence level. A situation with Boolean reasoning results is presented in formula (3).

$$S(X) = \begin{cases} true, \sum_{i=1}^{N} w_i * contr_{S,i}(x_i) \geq threshold \\ false, otherwise \end{cases} \quad (3)$$

In formula (3) the contribution function is defined according to formula (2). If only Boolean values are acceptable as a reasoning result, then undefined confidence level of a situation is usually counted as non-occurrence (which is implied by formula (3)). Otherwise, undefined confidence level usually results in undefined reasoning result.

Sometimes suitable weights, contributions and threshold can be defined by hand, but there also exist some techniques to facilitate setting the parameters of a situation. The techniques for choosing situation parameters include:

1. Defining the constraints on parameters. For some context states the fact of occurrence/non-occurrence of the situation is known. Those known states can be obtained by analyzing the subject area or by testing. Together they define a set of inequalities, which provides the limitations on parameters.
2. Voting for fixing option. If some test fails, there exist multiple possible methods to fix the situation parameters. For example, unexpected occurrence of the situation can be fixed by reducing the contribution of the triggered intervals, by redistributing the weights in favor of the context attributes with lower contribution on this testing sample or by increasing the threshold. Multiple testing results "vote" for taking or not taking different option to fix the parameters, and the option with good tradeoff between positive and negative votes is attempted first.
3. Learning situations from labelled data. The labeled data can be obtained by subject area analysis or by testing.

The choice of situation parameters is a large and relatively unexplored area, which mostly lies beyond the scope of this article. However, verification is a powerful way to prove that the parameters are defined correctly, or to identify the error.

It should also be noted that any situation can be transformed to have the any threshold between (0;1), and after the transformation the situation will be triggered at the same context state as before and remain well-defined (i.e. all the contributions are within [0;1], all the weights are within [0;1] and the weights sum up to 1). Those transformations are often used to achieve the same threshold for all the situations within the context space.

For example, one of the possible transformations is performed as follows. In order to change the threshold from the old value th_{old} to the new value th_{new} the developer can introduce an artificial context attribute, which has a single generic possible value (for example, non-numeric value *Stub*). The weight of the newly introduced context attribute is w_0. Other weights are multiplied by (1- w_0) for normalization. The contribution of the newly introduced context attribute is 1 if the threshold is increased ($th_{new} > th_{old}$) and 0 if the threshold is decreased ($th_{new} < th_{old}$). It can be straightforwardly proven that the proper value is $w_0 = \frac{th_{old} - th_{new}}{th_{old}}$ if the threshold is decreased ($0 < th_{new} < th_{old} \leq 1$) and $w_0 = \frac{th_{new} - th_{old}}{1 - th_{old}}$ if the threshold is increased ($0 \leq th_{old} < th_{new} < 1$). Note that it is enough to introduce a

single context attribute stub, and use it for all the situations that undergo the threshold modification.

In this article we view all the situations within the application space as having the same threshold. As we proved above, it can be considered without any loss of generality. Usually the threshold is defined for an application space, and then if any situation has different threshold, the transformation is applied.

In order to detect the confidence values of various situation relationships, CST is supplied with situation algebra concept. Operations, that constitute the basis of situation algebra, are presented in formula (4). The definitions comply with Zadeh operators [Za65].

$$\begin{aligned} &AND: (A \,\&\, B)(X) \;=\; min(A(X), B(X)) \\ &OR: (A \mid B)(X) \;=\; max(A(X), B(X)) \\ &NOT: (\neg A)(X) \;=\; 1 - A(X) \end{aligned} \tag{4}$$

Any arbitrary situation algebra expression can be evaluated, using the operations (4) as a basis. If a situation, provided as an argument for AND, OR or NOT operation, is undefined at some context state X, than the whole situation algebra expression is undefined at the context state X as well.

The situation awareness concepts, provided in the context spaces approach, have numerous benefits, including the following:

1. Integrated approach. CST contains the methods that lead reasoning process from raw sensory data up to the situation confidence interpretation.
2. Uncertainty integration. The situation reasoning can handle imprecision and even possible unavailability of sensor data.
3. Unified representation. Different situations might have different semantics. For example, situations can represent certain location, certain condition, certain activity. Context spaces theory allows defining and reasoning about the situations in a unified manner.
4. Clarity. Situations are human readable and can be relatively easily composed manually by the expert.

In the next sections we are going to provide an example scenario to clarify the concepts.

2.2 Context Spaces Approach Example

For the demonstration of the context spaces approach we developed an illustrative example. In the section 3.2 this example will be extended to the motivating scenario for the verification approach. We apply the theory of context spaces and its situation awareness capabilities in smart office environment.

Consider a smart office that monitors the conditions at the workplaces. A separate CST application space is associated with any arbitrary workplace. In that application space the situations are triggered if the confidence level reaches 0.7.

There are three context attributes of a particular interest: sensor measurement for light level (numerical), sensor measurement for noise level (numerical) and sensed light level switch (non-numeric On/Off). For simplicity we do not take into account possible sensor uncertainty or sensor unreliability.

Consider the CST situation *ConditionsAcceptable(X)*. The situation represents the fact that the light and noise levels at the workplace are acceptable. The situation

ConditionsAcceptable(X) is presented in the expression (5). We consider that the noise level and the luminance are of equal importance for evaluating the workplace conditions, so both weights are assigned at 0.5.

$$\begin{aligned}&\text{ConditionsAcceptable(X)} = \\ &\quad 0.5 * \text{contr}_{Light}(\text{LightLevel}) + 0.5 * contr_{\text{Noise}}(\text{NoiseLevel}) \\ &\text{contr}_{Light}(\text{LightLevel}) = \begin{cases} 0, \text{LightLevel} < 350 \\ 0.5, \text{LightLevel} \in [350, 500) \\ 1, \text{LightLevel} \geq 500 \end{cases} \\ &contr_{\text{Noise}}(\text{NoiseLevel}) = \begin{cases} 1, \text{NoiseLevel} \leq 40 \\ 0.7, \text{NoiseLevel} \in (40, 50] \\ 0.3, \text{NoiseLevel} \in (50, 60] \\ 0, \text{NoiseLevel} > 60 \end{cases}\end{aligned} \quad (5)$$

Consider also the situation *LightMalfunctions(X)*, which is presented in the expression (6). Light malfunction is detected, if the light switch is on, but still there is insufficient light at the workplace. The contribution of the light level is inversed comparing to the expression (5), because now the impact means unacceptability of the light level, not its acceptability (somewhat equivalent to the NOT operation from formula (4)). The contribution of the light switch position is straightforward – it has full impact if it is on, and if it is off it has no impact on the *LightMalfunctions(X)* situation.

As for the weights for the expression (6), the weight for the light level should not reach 0.7: otherwise with the luminance of lower than 350 lx the lamp will be counted as malfunctioning, but it could just be turned off. Both the position of the switch and insufficiency of the light are important in order to detect, so equal weights are chosen.

$$\begin{aligned}&\text{LightMalfunctions(X)} = 0.5 * \text{contr}_{Light}(\text{LightLevel}) + 0.5 * contr_{\text{Noise}}(\text{NoiseLevel}) \\ &\text{contr}_{Light}(\text{LightLevel}) = \begin{cases} 1, \text{LightLevel} < 350 \\ 0.5, \text{LightLevel} \in [350, 500) \\ 0, \text{LightLevel} \geq 500 \end{cases} \\ &contr_{\text{Switch}}(\text{SwitchPosition}) = \begin{cases} 1, \text{SwitchPosition} \in \{On\} \\ 0, \text{SwitchPosition} \in \{Off\} \end{cases}\end{aligned} \quad (6)$$

We are going to refer to this example throughout the article. In section 3 this example will be expanded to the motivating scenario for the verification approach. But in order to do that, we need to provide some additional definitions and formalize the verification concept.

2.3. Additional Definitions

In order to proceed further, we propose several more definitions and formally present the entities we are going to work with.

Let $\mathbb{C}$ be the set of all possible confidence values that can be returned by any situation space after reasoning. The formal definition of the set $\mathbb{C}$ is presented in formula (7).

$$\mathbb{C} \equiv \mathbb{R} \cup \{UD\} \quad (7)$$

A confidence value is a real value that numerically represents the confidence in the fact that a situation is occurring. In formula (7) *UD* (*undefined*) is a special value that shows

that confidence of the situation cannot be calculated. Usually confidence level, if defined, falls within the range [0;1], but in this article we will not restrict those boundaries.

Two confidence values $c_1 \epsilon \mathbb{C}$ and $c_2 \epsilon \mathbb{C}$ are equal if and only if they either both have the same numeric value, or they are both undefined. Inequalities are considered only for numeric confidence values. Any inequality between confidence values holds false, if there is *UD* on either side of it.

Let the set of all possible context states be $\mathbb{St}$. Therefore, an expression like $X \in \mathbb{St}$ merely means that X is a context state.

An arbitrary function *f*, that takes context state as an input and outputs a confidence level, will be referred to as a *situation*. A situation can be formally defined as $f: \mathbb{St} \rightarrow \mathbb{C}$. Ye et. al. [YD12] define a situation as "*external semantic interpretation* of sensor data"[YD12], where the *interpretation* means "situation assigns meaning to sensor data"[YD12] and *external* means "from the perspective of applications, rather than from sensors"[YD12]. Our general concept of a situation can be viewed as an application of that definition to the CST context model – the situation interprets low-level context information in a meaningful manner, and the rules of interpretation are given externally (by the expert) from an application perspective.

If for two situations f_1 and f_2 the expression (8) holds true (i.e. if for any context state X situations $f_1(X)$ and $f_2(X)$ produce the same output confidence value), we consider that situation f_1 is a representation of f_2 (or, symmetrically, f_2 is a representation of f_1). The situations f_1 and f_2 are considered to be different representations of the same situation.

$$\forall X \epsilon \mathbb{St}, \quad f_1(X) = f_2(X) \tag{8}$$

Obviously, any situation is a representation of itself (it directly follows from the definition).

To summarize, the term *situation* is used for any function that takes a context state as input and produces a confidence level as the output. The terms *situation space* or *CST situation* are used for the situations that can be represented in terms of the CST definition (expressions (1) and (2)). It means that any situation algebra expression can be called a situation, but it is not necessarily a situation space.

In order to supply CST with verification capabilities, we also need to introduce the definition of an *empty situation*. Situation S is *empty with respect to threshold f* if and only if there exists no context state, for which the confidence level of the situation S reaches *f*. Formally the definition is represented in formula (9).

$$S \text{ is empty w.r.t. } f \leftrightarrow \nexists X \in \mathbb{St}, \; S(X) \geq f \tag{9}$$

For example, the situation *(AtHome & ¬AtHome)(X)* is empty w.r.t. any threshold greater than 0.5. The concept of an empty situation will be of much practical value for the task of situation relations verification that will be introduced in the next section.

3. Situation Relations Verification in CST

3.1 Formal Verification by Emptiness Check

In this section we will justify the need to verify situation definitions, identify the challenges of that task and propose the general solution direction.

The methods to identify the situation can be classified into two groups [YD12]: learning-based approaches (definition by a set of examples, using supervised or unsupervised learning) and specification-based approaches (manual definition by an expert). Specification-based approaches do not require any training data beforehand and they often feature clearer situation representation and easier reasoning. On the other hand, learning-based approaches do not require preliminary manual situation definition (and therefore avoid most definition errors) and can automatically identify possible situations of interest that were not taken into account manually.

Context spaces theory follows specification-based approach. Situations in context spaces theory are defined manually, and the concept of the situation space is optimized to make the situations human-readable and easy to compose. Still the process of situation composition is prone to errors, and it will be highly beneficial if the user could formally verify, whether the defined situations and situation relations comply with certain properties.

Ye et. al. [YD12] identified multiple possible relationships between situations. Here we analyze the application of those relations to CST situation spaces. Temporal properties are out of scope of our article, so any relationships that involve timing or sequence of occurrence are intentionally left out.

1. **Generalization.** The occurrence of less general situation implies the occurrence of more general situation. For example, the situation *Driving* implies the situation *InTheCar*, which is more general.

In context spaces theory the generalization relations can be defined in a following manner (expression (10)).

$$\forall X \epsilon \mathbb{St},\ LessGeneralSituation(X) \rightarrow MoreGeneralSituation(X) \quad (10)$$

Using the situation algebra definitions (4) as a basis, the expression (10) can be rewritten as expression (11), and then converted to the expression (12).

$$\forall X \epsilon \mathbb{St},\ \neg(\ LessGeneralSituation(X) \& (\neg MoreGeneralSituation(X))) \quad (11)$$

$$\nexists X \epsilon \mathbb{St},\ LessGeneralSituation(X) \& (\neg MoreGeneralSituation(X)) \quad (12)$$

The expression (12) means that the situation $LessGeneralSituation(X) \& (\neg MoreGeneralSituation(X)$ should never occur, i.e. that the situation $LessGeneralSituation(X) \& (\neg MoreGeneralSituation(X)$ should be empty. The exact definition of an empty situation is provided in expression (9) in section 2.3.

So, the task of verifying generalization relationship was reduced to the task of checking the emptiness of a situation algebra expression.

2. **Composition**. Some situations can be decomposed into sub-situations. For example, the situation *AtHome* can be decomposed into the situations like *InTheLivingRoom*, *InTheKitchen*, *InTheBathroom*. For the context spaces theory it might be

formalized either as expression (13) or as expression (14).

$$\forall X \epsilon \mathbb{St}, \quad ComposedSituation(X) \rightarrow \\ Component1(X) \mid Component2(X) \mid \ldots \mid ComponentN(X) \tag{13}$$

$$\forall X \epsilon \mathbb{St}, \quad ComposedSituation(X) \leftrightarrow \\ Component1(X) \mid Component2(X) \mid \ldots \mid ComponentN(X) \tag{14}$$

The expression (14) implies that all particular sub-cases of the situation *ComposedSituation* do belong to at least one of the components, while the expression (13) does not have that assumption.

The expressions (13) and (14) can be rewritten as the expressions (15) and (16) respectively:

$$\nexists X \epsilon \mathbb{St}, ComposedSituation(X) \& \neg Component1(X) \& \neg Component2(X) \ldots \& \neg ComponentN(X) \tag{15}$$

$$\nexists X \epsilon \mathbb{St}, \ (ComposedSituation(X) \& \neg Component1(X) \& \neg Component2(X) \ldots \& \neg ComponentN(X)) \mid \\ (\neg ComposedSituation(X) \& Component1(X) \& Component2(X) \ldots \& ComponentN(X)) \tag{16}$$

Both expressions (15) and (16) can be viewed as an emptiness check task. It means that the task of verifying composition relationship can also be represented as a task of emptiness check.

3. **Dependence**. Ye et. al. [YD12] have concluded that "A situation *depends* on another situation if the occurrence of the former situation is determined by the occurrence of the latter situation" [YD12]. In terms of the context spaces theory it can be presented in the form of the expression (17).

$$\forall X \epsilon \mathbb{St}, \quad DependsOnSit(X) \leftarrow Sit(X) \tag{17}$$

The expression (17) can be rewritten as the expression (18), which in turn can be viewed as an emptiness check task.

$$\nexists X \epsilon \mathbb{St}, \quad (\neg DependsOnSit(X)) \& Sit(X) \tag{18}$$

So, the task of verifying the dependence can be represented as the task of situation emptiness check as well.

4. **Contradiction.** Contradicting situations should not occur at the same time. For example, the situation *Running* should not co-occur with the situation *Sitting*. The contradiction relationship for two generic situations is presented in the expression (19). The contradiction relations between multiple situations can be viewed as multiple contradictions between every pair of situations (i.e. every involved situation contradicts every other).

$$\nexists X \epsilon \mathbb{St}, \quad Sit1(X) \ \& \ Sit2(X) \tag{19}$$

The expression (19) shows that the test for contradiction can also be viewed as emptiness check.

It should be noted that the same kinds of relationships can apply not only to the single situations, but to the situation expressions as well. For example, the relationship *(InTheCar & Moving)<=>(Driving | CarPassenger)* can be viewed as slightly more complicated case of composition relationship: the joint situation *(InTheCar & Moving)* is composed of the sub-situations *Driving* and *CarPassenger*.

To summarize, if the expected relationship is represented as a situation algebra expression that should never hold true, then it is ready to be an input for the verification process (which, as we will show further, has emptiness check as its essential part). If the property under verification is represented as a situation algebra expression that should always hold true, then it can be converted to another format using the relationship (20).

$$\forall X \epsilon \mathbb{St},\ \textit{Expression (X)} \quad <=> \quad \nexists X \epsilon \mathbb{St},\ \neg \textit{(Expression (X))} \tag{20}$$

As a result, the analysis of possible situation relations, presented in this section, implies an important conclusion: **formal verification of situation relationships can be viewed as an emptiness check of a situation algebra expression**. If the task of emptiness check is solved for any arbitrary situation algebra expression, it will allow deriving a solution for the verification task.

3.2 Motivating Example

In order to demonstrate the functionality of the approach, we developed the following illustrative example.

Consider the smart office scenario presented in the section 2.2. The smart office is aware of two situations: *ConditionsAcceptable*, which implies that the light and noise conditions are acceptable for work, and *LightMalfunctions*, which says that despite the light is on, the illuminance level is still insufficient. So, *LightMalfunctions(X)* implies that the light level is insufficient to resume the work. In turn, light level insufficiency means that the conditions at the workplace are not acceptable. So, if the situation spaces are defined correctly, the situations *LightMalfunctions(X)* and *ConditionsAcceptable(X)* should not co-occur. There is a contradiction relationship between those situations. The formalization of that relation is presented in the expression (21).

$$\nexists X \epsilon \mathbb{St},\ \ LightMalfunctions(X) \& ConditionsAcceptable(X) \tag{21}$$

According to the application space definition, the threshold 0.7 is used to identify the occurrence, so in the expression (21) the situation is considered to be occurring if its confidence level reaches 0.7. The aim is to verify the relations between *LightMalfunctions(X)* and *ConditionsAcceptable(X)* and, therefore, check for emptiness the situation $(LightMalfunctions \& ConditionsAcceptable)(X)$ with respect to threshold 0.7.

In the next section we are going to discuss the solution approach for the emptiness check problem. This motivating scenario will be used as an illustration throughout the article.

4 Orthotope-based Situation Representation

In the section 3.1 we derived a conclusion, that in order to verify the situation relationships, we need to develop an efficient algorithm to check the emptiness of an arbitrary situation algebra expression. However, direct application of the situation algebra (formula (4)) allows reasoning only about the confidence level for a particular context state. It does not allow checking, whether certain condition holds for every possible context state.

As a solution approach we chose to enhance the situation representation for context spaces theory with a new situation format that will be able to represent any particular situation algebra expression as a situation and have a tractable algorithm for emptiness

check. Therefore, we introduced the following new situation space type – orthotope-based situation space.

In order to provide clear understanding of orthotope-based situation spaces concept we are going to refer to the example presented in the sections 2.2 and 3.2. The confidence level of *LightMalfunctions(X)* can be plotted using context attributes LightLevel and SwitchPosition as axes. The resulting plot is presented in the figure 1. The formal definition of the situation *LightMalfunctions(X)* and the corresponding values are derived from formula (6).

In the figure 1 it is clearly visible that the confidence levels are constant, if the context state is inside a certain combination of intervals over context attributes. Actually, this fact is true for any arbitrary situation, and it follows directly from formulas (1) and (2). Straightforward formalization of the figure 1 allows formulating the situation *LightMalfunctions(X)* as described in the expression (22).

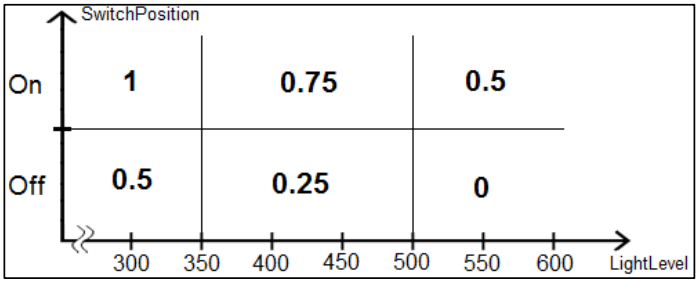


Fig. 1. Confidence level of *LightMalfunctions(X)*

$$\text{LightMalfunctions(X)} = \begin{bmatrix} 0.5,(\text{LightLevel}<350) \wedge (\text{SwitchPosition} \in \{\text{Off}\}) \\ 0.25,(\text{LightLevel} \in [350,500)) \wedge (\text{SwitchPosition} \in \{\text{Off}\}) \\ 0,(\text{LightLevel} \geq 500) \wedge (\text{SwitchPosition} \in \{\text{Off}\}) \\ 1,(\text{LightLevel}<350) \wedge (\text{SwitchPosition} \in \{\text{On}\}) \\ 0.75,(\text{LightLevel} \in [350,500)) \wedge (\text{SwitchPosition} \in \{\text{On}\}) \\ 0.5,(\text{LightLevel} \geq 500) \wedge (\text{SwitchPosition} \in \{\text{On}\}) \end{bmatrix} \quad (22)$$

Each condition in the expression (22) is a Cartesian product of generalized intervals (the concept of a generalized interval was presented in the section 2.1), i.e. a generalized orthotope [Co73].

The expression (22) provides representation of *LightMalfunctions(X)* as an orthotope-based situation space. General orthotope-based situation space can be defined according to the formula (23).

$$S(X) = \begin{bmatrix} a_1,(x_1 \in I_{1,1}) \wedge (x_2 \in I_{2,1}) \ldots \wedge (x_N \in I_{N,1}) \\ a_2,(x_1 \in I_{1,2}) \wedge (x_2 \in I_{2,1}) \ldots \wedge (x_N \in I_{N,1}) \\ \ldots \\ a_{r_1},(x_1 \in I_{1,r_1}) \wedge (x_2 \in I_{2,1}) \ldots \wedge (x_N \in I_{N,1}) \\ a_{r_1+1},(x_1 \in I_{1,1}) \wedge (x_2 \in I_{2,2}) \ldots \wedge (x_N \in I_{N,1}) \\ \ldots \\ a_L,(x_1 \in I_{1,r_1}) \wedge (x_2 \in I_{2,r_2}) \wedge \ldots \wedge (x_N \in I_{N,r_N}) \end{bmatrix} \quad (23)$$

In the formula (23) $I_{i,j}$ represents the j-th generalized interval over i-th context state. For i-th involved context attribute there are in total r_i non-overlapping intervals (numbered from

$I_{i,1}$ to $I_{i,ri}$), which cover the entire range of possible context attribute values. The number of context attributes, involved in a situation is referred to as N. The total number of the involved orthotopes (rows in the formula (23)) is referred to as $L = \prod_{i=1}^{N} r_i$. The total number of involved intervals is referred to as $R=\sum_{i=1}^{N} r_i$. The symbol ∧ refers to the conjunction (the symbol was chosen in order to avoid the confusion with situation algebra AND).

Every row inside the formula (23) defines a condition as a Cartesian product of multiple generalized intervals over N context attributes, i.e. an orthotope [Co73] in the application space. Therefore, every row of the formula (23) is referred to as an orthotope, and the situation itself is referred to as an orthotope-based situation representation.

For the situation *LightMalfunctions* from the example scenario (section 3.2), the number of involved context attributes is N=2 (*LightLevel* and *SwitchPosition*). Let *LightLevel* and *SwitchPositions* be the context attributes number 1 and 2 respectively. Therefore, the number of intervals for context attributes, $r_1 = 3$ (LightLevel<350; LightLevel ∈ [350;500) and LightLevel≥500) and $r_2 = 2$ (SwitchPosition ∈ {On} and SwitchPosition ∈ {Off}). The number of orthotopes is *L*=6 and the total number of intervals is *R*=5.

In subsequent sections we are going to prove several important properties of the orthotope-based situation space. In order to do that, we need to prove additional lemmas.

Lemma 4.1. Any particular context state belongs to some orthotope of the orthotope-based situation space.

Proof. Consider an arbitrary orthotope-based situation space S(X), defined over context attribute $CA_1 \ldots CA_N$. For context attribute CA_i the sets of intervals is $I(i,1) \ldots I(i,r_i)$. Consider an arbitrary particular context state X.

Consider the context attribute CA_i from the set $CA_1 \ldots CA_N$. Let's define the value for context attribute CA_i within context state X as x_i. The value x_i can as well be *undefined*. By definition the set of intervals $I(i,1) \ldots I(i,r_i)$ cover all possible set of context attribute values, i.e. any particular context attribute value belongs to some interval of that set. It applies to x_i as well. Let's define the interval x_i belongs to as $I(i,p_i)$.

To summarize: $(x_1 \in I(1,p_1)) \wedge (x_2 \in I(2,p_2)) \ldots \wedge (x_N \in I(N,p_N))$. By definition of orthotope-based situation space, all the combinations of intervals for different context attributes have the corresponding orthotope in the situation (formula (10)). It applies as well to $(x_1 \in I(1,p_1)) \wedge (x_2 \in I(2,p_2)) \ldots \wedge (x_N \in I(N,p_N))$. And that is the orthotope that context state X belongs to.

So for any arbitrary orthotope-based situation space and for any arbitrary particular context state X it was proven that it belongs to some orthotope of the orthotope-based situation space.

Q.E.D.■

Orthotope-based situation space is the key concept for verification. In the next section we are going to propose the verification algorithm using orthotope-based situation spaces as a useful intermediate representation.

5 Orthotope-based Situation Spaces for Situation Relations Verification

In this section we are going to propose and prove the approach to the formal verification of an arbitrary situation relation.

In general, verification can be performed as follows. In the subsections of this section we are going to discuss each step of this algorithm in details.

Step 1. Represent the property under verification as a situation algebra expression that should be checked for emptiness. This step was discussed in section 3, but, as will follow in section 5.4, the applicability of verification is not restricted to the properties described in 3.1.

Step 2. Convert the involved situations to orthotope-based situation spaces. Section 5.1 proposes the conversion algorithm from an original CST situation space to an orthotope-based situation space and discusses the algorithm correctness. The practical evaluation of the algorithm is provided in the section 6.1.

Step 3. Using the situation algebra expression under test and the converted input situations, derive the orthotope-based representation of the verified expression. Section 5.2 proposes the algorithm to derive the representation of the expression. Complexity of the proposed algorithm is defined in the section 6.2.

Step 4. Check orthotope-based representation for emptiness, and, if necessary, find the counterexamples. Section 5.3 proposes the emptiness check algorithm and discusses the algorithm correctness. The complexity of that algorithm is evaluated in the section 6.3. Section 5.3 also addresses the problem of counterexamples and proposes possible solutions.

As a result, the algorithm determines whether the expression under verification is empty (according to the definition (9)).

Section 5.4 summarizes the results of the sections 5.1-5.3, summarizes the integrated verification approach and discusses advanced practical aspects of the verification.

5.1 Conversion to an Orthotope-based Situation Space

The conversion from original CST situation space to orthotope-based situation space can be performed in a manner, described in the algorithm 5.1. In order to provide clear explanation of the algorithm, we need to propose the following lemma.

Lemma 5.1. Premise. Consider an arbitrary original CST situation space *sit(X)*, defined over N context attributes $CA_1 \ldots CA_N$ according to the formula (24).

$$sit(X) = \sum_{i=1}^{N} w_i \ *contr_i(x_i) \quad ; \text{contr}_i(x_i) = \begin{cases} a(i,1), x_i \in I(i,1) \\ a(i,2), x_i \in I(i,2) \\ \ldots \\ a(i,r(i)), x_i \in I(i,r(i)) \end{cases} \tag{24}$$

In formula (24) the weight of context attribute CA_i is referred to as w_i. Within the arbitrary context state X the value of context attribute CA_i is referred to as x_i. The number of involved intervals over context attribute CA_i is r(i). According to the definition of original CST situation space (section 2.1), the intervals over every involved context attribute cover the entire set of possible values of that context attribute, and do not overlap. It means, any particular value x_i of context attribute CA_i (i=1..n) does belong to one and only one interval in the set $I(i,1) \ldots I(i,r(i))$. Any contribution value a(i, j) can as well be undefined. If the contribution is undefined, then any sum involving that contribution will result in undefined confidence level.

Consider also a situation space *orthotope(X)* that is designed in a following manner (expression (25)). In expression (25) if the sum on any of the rows contains at least one *undefined* summand then the whole sum is *undefined* as well for that row.

$$orthotope(X) = \begin{cases} \sum_{i=1}^{N} w_i * a(i,k_i), (x_1 \in I(1,k_1)) \wedge \left(x_2 \in I(2,k_2)\right) \wedge \ldots \wedge (x_N \in I(N,k_N)); \\ k_1 = 1,\ k_2 = 1, \ldots, k_N = 1 \\ \sum_{i=1}^{N} w_i * a(i,k_i), (x_1 \in I(1,k_1)) \wedge \left(x_2 \in I(2,k_2)\right) \wedge \ldots \wedge (x_N \in I(N,k_N)); \\ k_1 = 2,\ k_2 = 1, \ldots, k_N = 1 \\ \ldots \\ \sum_{i=1}^{N} w_i * a(i,k_i), (x_1 \in I(1,k_1)) \wedge \left(x_2 \in I(2,k_2)\right) \wedge \ldots \wedge (x_N \in I(N,k_N)); \\ k_1 = r(1),\ k_2 = 1, \ldots, k_N = 1 \\ \sum_{i=1}^{N} w_i * a(i,k_i), (x_1 \in I(1,k_1)) \wedge \left(x_2 \in I(2,k_2)\right) \wedge \ldots \wedge (x_N \in I(N,k_N)); \\ k_1 = 1,\ k_2 = 2, \ldots, k_N = 1 \\ \ldots \\ \sum_{i=1}^{N} w_i * a(i,k_i), (x_1 \in I(1,k_1)) \wedge \left(x_2 \in I(2,k_2)\right) \wedge \ldots \wedge (x_N \in I(N,k_N)); \\ k_1 = r(1),\ k_2 = r(2), \ldots, k_N = r(N) \end{cases} \quad (25)$$

Actually, the orthotopes of situation (25) are obtained using brute-force iteration through every possible Cartesian product of intervals, mentioned in situation (24).

Lemma statements: 1) *orthotope(X)* is an orthotope-based situation space.
2) *orthotope(X)* and *sit(X)* represent the same situation.

Proof:

We can prove the statement 1 as follows. Formula (25) is compliant with the definition formula (22) - every i-th context attribute is divided into the set of intervals I(i,1)…I(I,r(i)), and by construction the situation space *orthotope(X)* assigns the confidence level to every combination of those intervals. Every set of intervals I(i,1)…I(i,r(i)) (i=1..N) over context attribute CA_i covers the entire set of possible context attribute values and does not overlap within each other – both those facts are the part of the definition of original CST situation space *Sit(X)*. All those facts combined make the definition of *orthotope(X)* fully compliant with the definition of an orthotope-based situation space provided in the section 4. It means, that *orthotope(X)* is an orthotope-based situation space. **Q.E.D. for statement 1.**

Statement 2 can be proven as follows. By definition the situation *orthotope(X)* is a representation of situation *sit(X)*, if for any arbitrary particular context state X the confidence levels of the situation spaces *sit(X)* and *orthotope(X)* are equal.

Consider a random particular context state X. Consider an arbitrary context attribute CA_i from the set $CA_1 \ldots CA_N$. The value of context attribute CA_i in the context state X is referred to as x_i. According to the definition of *sit(X)*, the set of intervals, I(i,1)…I(i,.r(i)) covers the entire context attribute CA_i and do not overlap, i.e. any particular context attribute value belongs to one and only one of the intervals. So, x_i belongs to one of the intervals I(i,1)…I(i,.r(i)). That interval will be referred as $I(i,p_i)$.

So, $x_i \in I(i,p_i)$ and that applies to any context attribute CA_i within the set of $CA_1 \ldots CA_N$. The results for all the involved context attributes are summarized in the expression (26).

$$X: (x_1 \in I(1,p_1)) \wedge \left(x_2 \in I(2,p_2)\right) \wedge \ldots \wedge (x_N \in I(N,p_N)) \quad (26)$$

The expression (26) identifies one of the orthotopes in *orthotope(X)* according to the expression (25). In particular, it is the orthotope where $k_1=p_1$, $k_2=p_2,\ldots,k_N=p_N$. So, according to the expression (25) the confidence value of *orthotope(X)* at state X is $orthotope(X) = \sum_{i=1}^{N} w_i * a(i,p_i)$.

The context attribute value x_i belongs to the interval $I(i,p_i)$, and therefore according to the expression (24) the contribution of the context attribute CA_i is $a(i,p_i)$ (i=1..N). The total confidence level for *sit(X)* is a weighted sum of contributions, so according to the formula (24) $sit(X) = \sum_{i=1}^{N} w_i * a(i,p_i)$.

To summarize, $orthotope(X) = \sum_{i=1}^{N} w_i * a(i, p_i) = sit(X)$. If any of the $a(i,p_i)$ is undefined, then the confidence level will be undefined for both of the situations, and the results will remain equal..

As a result, for an arbitrary context state X the reasoning result of *orthotope(X)* is the same as reasoning result of *sit(X)*. It means that *orthotope(X)* and *sit(X)* represent the same situation.

Q.E.D. for statement 2. ■

The situation *sit(X)* is an arbitrary original CST situation, and for any *sit(X)* the orthotope-based representation *orthotope(X)* can be composed. Therefore, lemma 5.1 directly implies that for any original CST situation space there exists an orthotope-based representation.

Lemma 5.1 also proposes the equivalent representation of an arbitrary original CST situation space in an orthotope-based format. Therefore, any algorithm that takes a situation like expression (24) (and that can be any original CST situation) as an input and provides a situation like expression (25) as an output, is a correct algorithm: it takes arbitrary original CST situation space as an input and returns orthotope-based (according to lemma 5.1. statement 1) representation of the same situation (according to lemma 5.1. statement 2) as an output, and that is what we expect from a conversion algorithm.

For example, orthotope-based situation can be built by composing orthotope after orthotope (i.e. row after row in formula (25)) using the algorithm 5.1.

Algorithm 5.1. Input. Any arbitrary original CST situation *sit(X)*, defined according to the formula (24).

Algorithm pseudocode:

//Consitruction the situation from formula (25) row after row
SituationSpace orthotope = new SituationSpace(); //Creating new situation space
for every combination $k_1,k_2,\ldots,k_N$ where $k_1 = 1..r(1)$, $k_2 = 1..r(2)$,...,$k_N=1..r(N)$
  //Start constructing the new orthotope
  OrthotopeDescription oDescription = new OrthotopeDescription();
  ConfidenceLevel confidence = 0;
  //Creating the orthotope and confidence – one context attribute after another
  for j=1..N
    orthotopeDescription.addContextAttributeInterval(CA_j, $I(j,k_j)$);
    *confidence += w_j*a(j,k_j);*
  end for
  orthotope.addOrthotope(oDescription, confidence);
end for

Output. Situation *orthotope(X)*.

The complexity of the algorithm 5.1 is evaluated in section 6.

Consider an example of the algorithm 5.1, applied to the situation *LightMalfunctions(X)* (expression (6)) from the sample scenario presented in section 3.2. There are 3 involved intervals for the light level (LightLevel<350; LightLevel ∈ [350;500) and LightLevel≥500) and 2 involved intervals for the switch position (SwitchPosition ∈ {On} and SwitchPosition ∈ {Off}). The possible combinations of intervals and the corresponding confidence levels are provided in the expression (27).

(LightLevel<350) ^ (SwitchPosition ∈ {Off}): $0.5 * 1 + 0.5 * 0 = 0.5$
(LightLevel ∈ [350,500)) ^ (SwitchPosition ∈ {Off}): 0.5*0.5+0.5*0 = 0.25
(LightLevel≥500) ^ (SwitchPosition ∈ {Off}): 0.5*0+0.5*0=0 (27)

(LightLevel<350) ^ (SwitchPosition ∈ {On}): $0.5 * 1 + 0.5 * 1 = 1$
(LightLevel ∈ [350,500)) ^ (SwitchPosition ∈ {On}): 0.5*0.5+0.5*1 = 0.75
(LightLevel≥500) ^ (SwitchPosition ∈ {On}): $0.5 * 0 + 0.5 * 1 = 0.5$

The result of the transformation of the situation *LightMalfunctions(X)* is presented in the expression (22). Using the similar methods, the situation *ConditionAcceptable(X)* can be represented in an orthotope-based situation format in a manner described in expression (28).

$$ConditionsAcceptable(X) = \begin{bmatrix} 0.5, (LightLevel < 350) \wedge (NoiseLevel \leq 40) \\ 0.35, (LightLevel < 350) \wedge (NoiseLevel \in [40,50)) \\ 0.15, (LightLevel < 350) \wedge (NoiseLevel \in [50,60)) \\ 0, (LightLevel < 350) \wedge (NoiseLevel > 60) \\ 0.75, \big(LightLevel \in [350,500)\big) \wedge (NoiseLevel \leq 40) \\ 0.6, \big(LightLevel \in [350,500)\big) \wedge (NoiseLevel \in [40,50)) \\ 0.4, \big(LightLevel \in [350,500)\big) \wedge (NoiseLevel \in [50,60)) \\ 0.25, \big(LightLevel \in [350,500)\big) \wedge (NoiseLevel > 60) \\ 1, (LightLevel \geq 500) \wedge (NoiseLevel \leq 40) \\ 0.85, (LightLevel \geq 500) \wedge (NoiseLevel \in [40,50)) \\ 0.65, (LightLevel \geq 500) \wedge (NoiseLevel \in [50,60)) \\ 0.5, (LightLevel \geq 500) \wedge (NoiseLevel > 60) \end{bmatrix} \quad (28)$$

In the next section we demonstrate that any arbitrary situation algebra expression over the orthotope-based situation spaces can be represented as an orthotope-based situation space. The orthotope-based situations *LightMalfunction(X)* (expression (22)) and *ConditionsAcceptable(X)* (expression (28)) will be used for illustration purposes.

5.2 Closure under Situation Algebra

In order to derive the expected conclusion about the closure under situation algebra, several additional lemmas are required. Lemma 5.2.1 provides the method to preprocess the involved situations properly. Lemma 5.2.2 facilitates the new situation composition. Lemma 5.2.3 provides the sufficient conditions for the closure under an operation for orthotope-based situation spaces. Lemma 5.2.4 proves the closure under any situation algebra expression (with certain requirements for the situation algebra basis), and concludes the closure proof. The algorithm 5.2 for deriving the orthotope-based situation representation of an arbitrary situation algebra expression emerges as a result of the proof.

Lemma 5.2.1. Premise.

Consider a function $a(l_1,l_2,\ldots,l_N)$, that accepts N integer arguments and returns a confidence level. Any input argument l_i can have a value within the range $[1;r_i]$.

Consider an arbitrary set of context attributes $CA_1 \ldots CA_{N+1}$. For every context attribute CA_i there is a set of intervals $I_{i,1} \ldots I_{i,\ ri}$ defined. Those intervals cover the entire set of possible values for context attribute CA_i and do not overlap.

Consider situation *A(X)*, defined by formula (29) over the context attributes $CA_1 \ldots CA_N$.

$$A(X) = \begin{cases} a(1,1,\dots,1), (x_1 \in I_{1,1}) \wedge (x_2 \in I_{2,1}) \dots \wedge (x_N \in I_{N,1}) \\ a(2,1,\dots,1), (x_1 \in I_{1,2}) \wedge (x_2 \in I_{2,1}) \dots \wedge (x_N \in I_{N,1}) \\ \dots \\ a(r_1,1,\dots,1), (x_1 \in I_{1,r_1}) \wedge (x_2 \in I_{2,1}) \dots \wedge (x_N \in I_{N,1}) \\ a(1,2,\dots,1), (x_1 \in I_{1,1}) \wedge (x_2 \in I_{2,2}) \dots \wedge (x_N \in I_{N,1}) \\ \dots \\ a(r_1,r_2,\dots,r_N), (x_1 \in I_{1,r_1}) \wedge (x_2 \in I_{2,r_2}) \wedge \dots \wedge (x_N \in I_{N,r_N}) \end{cases} \quad (29)$$

Consider the situation *B(X)*, defined according to formula (30) over the context attributes $CA_1 \dots CA_{N+1}$

$$B(X) = \begin{cases} a(1,1,\dots,1), (x_1 \in I_{1,1}) \wedge (x_2 \in I_{2,1}) \dots \wedge (x_N \in I_{N,1}) \wedge (x_{N+1} \in I_{N+1,1}) \\ a(2,1,\dots,1), (x_1 \in I_{1,2}) \wedge (x_2 \in I_{2,1}) \dots \wedge (x_N \in I_{N,1}) \wedge (x_{N+1} \in I_{N+1,1}) \\ \dots \\ a(r_1,1,\dots,1), (x_1 \in I_{1,r_1}) \wedge (x_2 \in I_{2,1}) \dots \wedge (x_N \in I_{N,1}) \wedge (x_{N+1} \in I_{N+1,1}) \\ a(1,2,\dots,1), (x_1 \in I_{1,1}) \wedge (x_2 \in I_{2,2}) \dots \wedge (x_N \in I_{N,1}) \wedge (x_{N+1} \in I_{N+1,1}) \\ \dots \\ a(r_1,r_2,\dots,r_N), (x_1 \in I_{1,r_1}) \wedge (x_2 \in I_{2,r_2}) \wedge \dots \wedge (x_N \in I_{N,r_N}) \wedge (x_{N+1} \in I_{N+1,1}) \\ a(1,1,\dots,1), (x_1 \in I_{1,1}) \wedge (x_2 \in I_{2,1}) \dots \wedge (x_N \in I_{N,1}) \wedge (x_{N+1} \in I_{N+1,2}) \\ \dots \\ a(r_1,r_2,\dots,r_N), (x_1 \in I_{1,r_1}) \wedge (x_2 \in I_{2,r_2}) \wedge \dots \wedge (x_N \in I_{N,r_N}) \wedge (x_{N+1} \in I_{N+1,2}) \\ a(1,1,\dots,1), (x_1 \in I_{1,1}) \wedge (x_2 \in I_{2,1}) \dots \wedge (x_N \in I_{N,1}) \wedge (x_{N+1} \in I_{N+1,3}) \\ \dots \\ a(r_1,r_2,\dots,r_N), (x_1 \in I_{1,r_1}) \wedge (x_2 \in I_{2,r_2}) \wedge \dots \wedge (x_N \in I_{N,r_N}) \wedge (x_{N+1} \in I_{N+1,r_{N+1}}) \end{cases} \quad (30)$$

So, situation B is defined over context attributes $CA_1 \dots CA_{N+1}$. The context attributes $CA_1 \dots CA_N$ are divided into the same intervals, as for situation A. The entire set of possible values for context attribute CA_{N+1} is decomposed into intervals $I_{N+1,1} \dots I_{N+1,r_{N+1}}$. As follows from formula (30) the confidence level of B does not depend on the CA_{N+1} context attribute value.

Lemma statements: 1) *A(X)* and *B(X)* are different representations of the same situation.

2) Both *A(X)* and *B(X)* are orthotope-based situation spaces.

Before the proof starts, consider some clarifications. Lemma 5.2.1 allows to derive more concise representations of the situations (use *A(X)* instead of *B(X)*), and to get rid of the context attributes that do not influence the confidence level. This transformation provides more concise and clear representation and reduces the efforts for situation reasoning.

For example, if by some calculations, the user finds out that the situation *NoiseLevelOK(X)* can be represented by formula (31) then the same situation *NoiseLevelOK(X)* can be represented in a simpler manner, by the expression (32) (for simplicity undefined context attributes are not considered in the example).

$$\text{NoiseLevelOK(X)}=\begin{bmatrix} 1,(\text{NoiseLevel} \leq 40) \wedge (\text{LightLevel} < 300) \\ 0.7,(\text{NoiseLevel} \in (40,50]) \wedge (\text{LightLevel} < 300) \\ 0.3,(\text{NoiseLevel} \in (50,60]) \wedge (\text{LightLevel} < 300) \\ 0,(\text{NoiseLevel}>60) \wedge (\text{LightLevel} < 300) \\ 1,(\text{NoiseLevel} \leq 40) \wedge (\text{LightLevel} \geq 300) \\ 0.7,(\text{NoiseLevel} \in (40,50]) \wedge (\text{LightLevel} \geq 300) \\ 0.3,(\text{NoiseLevel} \in (50,60]) \wedge (\text{LightLevel} \geq 300) \\ 0,(\text{NoiseLevel}>60) \wedge (\text{LightLevel} \geq 300) \end{bmatrix} \quad (31)$$

$$\text{NoiseLevelOK(X)}=\begin{bmatrix} 1,\text{NoiseLevel} \leq 40 \\ 0.7,\text{NoiseLevel} \in (40,50] \\ 0.3,\text{NoiseLevel} \in (50,60] \\ 0,\text{otherwise} \end{bmatrix} \quad (32)$$

Lemma 5.2.1 can also be used in another direction and introduce new context attributes into consideration, without altering the situation itself (use *B(X)* instead of *A(X)*). That transformation does not add any information, and might seem unnecessary complication at the first glance. However, the possibility of that transformation means that when working with a set of orthotope-based situation spaces, we can treat them as if they were all defined over the same set of context attributes. It will allow simplifying the subsequent algorithms. Whichever way the transformation proceeds, the statement 2 allows stating that the transformation result is an orthotope-based situation space.

Proof. Let's start with statement 2.

The definition of *A(X)* and *B(X)* is compliant with the formula (23). By definitions of *A(X)* and *B(X)* the set of intervals for every involved context attribute covers the entire possible set of context attribute values, and the intervals do not overlap with each other. According to the definitions in expressions (29) and (30) the corresponding confidence level is defined for every combination of intervals. Taken together, those facts imply that both *A(X)* and *B(X)* entirely comply with the definition provided in section 4, and therefore both *A(X)* and *B(X)* are orthotope-based situation spaces. **Q.E.D. for statement 2.**

Consider the proof for statement 1. The situations *A(X)* and *B(X)* are the representations of the same situation if and only if for any arbitrary particular context state X the confidence levels of *A(X)* and *B(X)* are equal.

Consider an arbitrary particular context state X. For any context state CA_i (i=1…N+1) context state X has particular context attribute value x_i (if the value for context state CA_i is missing from context state X, it will result just in having *undefined* as a value for x_i, which is the special case of particular context attribute value).

By definition of lemma the set of intervals $I_{i,1} \ldots I_{i,r_i}$ covers the entire range of possible values of context attribute CA_i (i=1…N+1). Also by definition the intervals $I_{i,1} \ldots I_{i,r_i}$ do not overlap. It means that context attribute value x_i belongs to one of those intervals. Let's refer to the number of that interval as k_i (and thus the interval itself is $I_{i,ki}$). Summarizing those facts for all the involved context attributes, allows deriving the confidence level *A(X)* using the formula (29) directly. Expression (33) presents the confidence value calculation.

$$\left.\begin{matrix} x_1 \in I_{1,k_1} \\ x_2 \in I_{2,k_2} \\ \ldots \\ x_N \in I_{N,K_N} \end{matrix}\right\} => A(\text{X}) = \text{a}(\text{k}_1, \text{k}_2, \ldots, \text{k}_\text{N}) \quad (33)$$

However, context states $CA_1 \ldots CA_N$ in the situation *B(X)* are divided using the same intervals as for the situation *A(X)* and those intervals are assigned the same numbers. It allows, in turn, calculating confidence value using the formula (30). The calculation process is presented in expression (34).

$$\left.\begin{array}{c} x_1 \in I_{1,k_1} \\ x_2 \in I_{2,k_2} \\ \ldots \\ x_N \in I_{N,K_N} \\ x_{N+1} \in I_{N+1,K_{N+1}} \end{array}\right\} => B(\mathrm{X}) = \mathrm{a}(\mathrm{k}_1, \mathrm{k}_2, \ldots, \mathrm{k}_\mathrm{N}) \quad (34)$$

Expressions (33) and (34) imply that $\mathrm{B(X)} = \mathrm{a}(\mathrm{k}_1, \mathrm{k}_2, \ldots, \mathrm{k}_\mathrm{N}) = \mathrm{A(X)}$.

So, for any arbitrary context state X, the confidence level *B(X)* is equal to the confidence level of *A(X)*, and by definition it means that *A(X)* and *B(X)* represent the same situation. **Q.E.D. for statement 1.**

Lemma 5.2.1 was derived in order to facilitate further proofs. However, the results of lemma 5.2.1 can be used separately in order to detect the unimportant context attributes and remove them from consideration without changing the situation itself.

Lemma 5.2.2 is an auxiliary lemma that proves some features of multiple interval intersections. Those features are used in subsequent proofs.

Lemma 5.2.2. Premise.

Consider an arbitrary context attribute CA.

Consider K sets of intervals (see expression (35)).

$$\begin{array}{l} Set_1: \{I(1,1), I(1,2), \ldots, I(1, r_1)\} \\ Set_2: \{I(2,1), I(2,2), \ldots, I(2, r_2)\} \\ \ldots \\ Set_K: \{I(K,1), I(K,2), \ldots, I(K, r_K)\} \end{array} \quad (35)$$

Every set of intervals $I(i,1), I(i,2), \ldots, I(i, r_i)$ (i=1..K) covers the entire set of possible context attribute values of CA, i.e. any particular value *x* of context attribute CA belongs to some interval of set Set_i. Intervals within one set do not overlap with each other.

Consider a new set of intervals *SetNew*, defined according to formula (36):

$$\{I(1,l_1) \cap I(2,l_2) \cap \ldots \cap I(K,l_K) \mid l_1 = 1..r_1, l_2 = 1..r_2, \ldots, l_K = 1..r_K\} \quad (36)$$

Consider also the set of intervals *SetNew2*. The set *SetNew2* is derived from the set *SetNew* by removing all the empty intervals.

Lemma statements:

1. *SetNew* covers all possible values of context attribute CA, i.e. every possible particular value *x* of context attribute CA belongs to at least one of the intervals.
2. The intervals within *SetNew* do not overlap, i.e. any particular value *x* of context attribute CA belongs to at most one interval.

Statements 1 and 2 together imply that the intervals from *SetNew* can be used for an orthotope-based situation space.

3. Statements 1 and 2, as well as their implication, hold true for *SetNew2* as well.

Proof:

Consider an arbitrary particular value *x* of context attribute CA.

By definition of the set Set_i (i=1..K) its intervals cover the entire sent of possible context attribute values and do not overlap, so the context attribute value *x* belongs to one interval of that set: either I(i,1), or I(i,2), …, or I(i,r_i). Let's refer to that interval as I(i,p_i).

Taking all the set together, context attribute value *x* belongs to all of those intervals I(1,p_1), I(2,p_2),…, I(K,p_K) (every p_i is in range [1;r_i]). And the value that belongs to several intervals at once, belongs to the intersection of those intervals.

So, $x \in I(1, p_1) \cap I(2, p_2) \cap ... \cap I(K, p_K)$. But by definition of *SetNew* the interval $I(1, p_1) \cap I(2, p_2) \cap ... \cap I(K, p_K)$ belongs to that set (it is the case when $l_1=p_1$, $l_2=p_2$, etc.). It proves that *x* belongs to one of the intervals for *SetNew*. So, any arbitrary particular context state *x* belongs to some interval of the set *SetNew* and it means that the set *SetNew* covers the entire set of possible values for context state CA. **Q.E.D. for statement 1.**

Consider arbitrary particular context attribute value *x*. That context attribute value belongs to some interval of *SetNew* according to statement 1: $x \in I(1, p_1) \cap I(2, p_2) \cap ... \cap I(K, p_K)$. Let's prove statement 2 by contradiction. Consider that there is another interval within the set *SetNew* and context attribute value *x* belong to that interval as well. Let it be the interval $I(1, m_1) \cap I(2, m_2) \cap ... \cap I(K, m_K)$, and for at least one index $m_j \neq p_j$. However, if the context state *x* belongs to the intersection of intervals $I(1, m_1) \cap I(2, m_2) \cap ... \cap I(K, m_K)$, it means that it belongs to any interval in that intersection (by definition, belonging to intersection means belonging to all the intervals). In particular, it means that $x \in I(j, m_j)$. But according to our pervious definitions $x \in I(j, p_j)$. By definition the intervals within the set Set_j do not intersect with each other, it means that if $m_j \neq p_j$, than the intersection of $I(j, p_j)$ and $I(j, m_j)$ is empty, i.e. there is no particular context attribute value that belongs to both of those intervals. However, the context attribute value *x* belongs to both of those intervals. It is a contradiction. The contradiction shows that the assumption was wrong, and *x* belongs to at most one of the intervals within the set *SetNew*. So, there is no context state that belongs to two or more intervals of *SetNew*, and it means that the intervals of that set are non-overlapping. **Q.E.D for statement 2.**

Let's proceed to the implication. The set *SetNew* decomposes the context attribute CA to the set of non-overlapping intervals (statement 2), that cover the entire set of possible context state values (statement 1). It is enough for compliance with the requirements for interval set of an orthotope-based situation space definition (see the definition in section 4).

It should be noted that many intervals of *SetNew* are likely to be empty. For an orthotope-based situation space it will just result in unreachable and therefore useless orthotopes, but the definition will still be consistent. However, the unreachable orthotopes will result in the waste of memory and the waste of processing time, and therefore unreachable orthotopes should be removed from the consideration, if possible.

Consider the proof for statement 3. Statement 3 contains 2 sub-statements:

Sub-statement 3.1. The statement 1 still holds true for the set *SetNew2*, which consists of all non-empty intervals of the set *SetNew*. Let's prove it by contradiction. It is already proven that statement 1 is true for *SetNew*. Assume that several empty intervals were removed, and after that the statement 1 is no longer true. It means that there exists particular context attribute value *x* that does not belong to any of the remaining intervals. However, before the intervals were removed from the *SetNew*, that context state did belong to some interval in that set (according to already proven statement 1). It implies that the context attribute value *x* did belong to one of the removed intervals. But the removed intervals are empty by definition of *SetNew2*, and no context state belongs to them. It is a contradiction. That means, the assumption is wrong and the intervals for *SetNew2* cover the

entire set of possible context attribute values. **Q.E.D for sub-statement 3.1.**

Sub-statement 3.2. The proof of statement 2 for the set *SetNew2* can be done in exactly the same manner as for *SetNew*. The fact that some intervals were removed from the set will not change anything in the proof. **Sub-statement 3.2. is proven.**

The proven sub-statements 3.1 and 3.2 show that the set *SetNew2* is suitable for orthotope-based situation space for exactly the same reasons as *SetNew*. And it proves the implication from statement 3.

As a result, all three statements and their implications are proven. The proof is complete. **Lemma 5.2.2 is proven.■**

The results of lemma 5.2.2 facilitate the proof of lemma 5.2.3. Lemma 5.2.3 is probably the most important lemma in the section 5. Orthotope-based situation spaces are closed under specific kinds of operations, and lemma 5.2.3 proposes the sufficient conditions for the operation, in order for an orthotope-based situation space to be closed under it. Later it can be relatively easily proven that any kind of situation algebra expressions over different kinds of basis functions do comply with that sufficient condition.

Lemma 5.2.3. Premise: Consider the function presented in formula (37).

$$func: \mathbb{C}^K \rightarrow \mathbb{C}, \quad func(c_1, c_2, \dots, c_K) = c \tag{37}$$

As presented in formula (37), the function *func* takes K arguments of confidence value type. The returned value is of a confidence type as well.

Consider a set of K arbitrary orthotope-based situation spaces: $Sit_1(X)$, $Sit_2(X)$, …, $Sit_K(X)$

Consider a situation *op(X)* defined in formula (38).

$$\forall X \epsilon \mathbb{S}\mathbb{t}, \quad op(X) = func(Sit_1(X), \dots, Sit_K(X)) \tag{38}$$

Lemma statement: For *op(X)* there exists an orthotope-based situation representation.

Proof.

Let's introduce a set of context attributes $CA_1...CA_N$. An arbitrary context attribute is added to that set, if it is mentioned in at least one of the situations Sit_1, Sit_2, …, Sit_K. According to lemma 5.2.1 without the loss of generality we can consider that every situation space from Sit_1, Sit_2, …, Sit_K set is defined over $CA_1...CA_N$. If situation space Sit_i does not involve some context attribute CA_j, that context attribute can be added into consideration using the transformation rules from lemma 5.2.1. Also according to lemma 5.2.1 after the transformation all the situations Sit_1, Sit_2, …, Sit_K will still retain the orthotope-based situation space format.

We are going to prove lemma 5.2.3 by construction, i.e. by showing and proving the algorithm that will derive orthotope-based representation of *op(X)*.

Let's introduce new situation space *orthotope(X)* in a following manner. The situation space *orthotope(X)* will be defined over context attributes $CA_1 \dots CA_N$. At first, for the situation space *orthotope(X)* let's introduce sets of intervals over the context attributes.

Consider the method to derive the set of intervals for arbitrary context attribute CA_i from the list of $CA_1 \dots CA_N$. For any context attribute from that list the procedure is the same. Let's refer to the number of intervals that situation $Sit_j(X)$ has over the context state CA_i as $r(i,j)$ and refer to the intervals themselves as $I(i,j,1) \dots I(i,j,r(i,j))$. That means, different situation spaces have the following intervals over context attribute CA_i (see expression (39)).

$$\begin{aligned} &Sit_1(X): I(i,1,1), I(i,1,2), \dots, I(i,1,r(i,1)) \\ &Sit_2(X): I(i,2,1), I(i,2,2), \dots, I(i,2,r(i,2)) \\ &\dots \\ &Sit_K(X): I(i,K,1), I(i,K,2), \dots, I(i,K,r(i,K)) \end{aligned} \quad (39)$$

Every set of intervals (i.e. every row of expression (39)) is an entire interval set of an orthotope-based situation space for a certain context attribute. So, every set is compliant with the definition of an orthotope-based situation space, and that means that any set of intervals from expression (39) covers the entire set of possible context state attributes, and does not have the overlaps. It means that lemma 5.2.2 is applicable. Let's construct the set of intervals, according to the lemma 5.2.2. The result is the expression (40).

$$\begin{aligned} \{I(i,1,l_1) \cap I(i,2,l_2) \cap \dots \cap I(i,K,l_K) \,|\, &l_1 = 1..r(i,1), l_2 = 1..r(i,2), \\ &\dots, l_K = 1..r(i,K)\} \end{aligned} \quad (40)$$

According to the lemma 5.2.2, the constructed set will divide the entire context attribute CA_i into the set of non-overlapping intervals. Let's consider only non-empty intervals from that set. According to lemma 5.2.2 statement 3 the set of intervals will still divide the entire context attribute CA_i into the set of non-overlapping intervals. Let the remaining number of intervals be $r'(i)$, and let's refer to those remaining intervals as $I'(i,1), I'(i,2), \dots, I'(i,r'(i))$.

It should be noted that $r'(i) \leq \prod_{j=1}^{K} r(i,j)$ – in a special case there will be no empty intersections, and the number of resulting intervals will be $\prod_{j=1}^{K} r(i,j)$ (all possible combinations), in other cases some intervals will be removed if they are empty.

So, applying the procedure from lemma 5.2.2 will allow dividing the set of all possible context attribute values of every involved context attribute to the set of non-overlapping intervals. Applying the same procedure for every context attribute $CA_1..CA_N$ will construct N sets of intervals over N context attributes. Each of those sets will be suitable for an orthotope-based situation space (according to the implication of statement 3 from lemma 5.2.2). So, taken together they will construct the structure of orthotopes for an orthotope-based situation space *orthotope(X)*.

Before advancing further, we need to derive a sub-statement 5.2.3.1.

Sub-statement 5.2.3.1.: every orthotope of the *orthotope(X)* is reachable, i.e. for every orthotope there exists a particular context state that belongs to it. The proof is following. If the orthotope consists of non-empty intervals for all involved context attributes, then the orthotope is reachable - context state can be composed by taking arbitrary particular context attribute value within the interval (and taking the particular value within the interval is possible, as the intervals are non-empty, i.e. there exist a value or values that belong to the interval). It means that if the orthotope is completely unreachable, that orthotope contains at least one empty interval for a context attribute. However, there are no empty intervals involved for any context attribute in *orthotope(X)* – they all were removed according to the rules provided in statement 3 of lemma 5.2.2. It means that all the orthotopes of *orthotope(X)* are reachable. **Q.E.D. for sub-statement 5.2.3.1.**

In order to complete the definition of *orthotope(X)*, for every orthotope in the situation the confidence level should be defined. Let's do it in a following manner: for every orthotope take an arbitrary particular context state that belongs to it. At least one such state

does exist according to the sub-statement 5.2.3.1, so it is possible to do so. Let it be the context state X'. Then the confidence level of that orthotope should be defined as $op(X') = func(Sit_1(X'), \dots, Sit_K(X'))$.

As a result, we defined an orthotope-based situation space *orthotope(X)* in a specific manner. Now let's prove that *orthotope(X)* and *op(X)* are different representations of the same situation. If they are, that will complete the proof.

By definition two situations are the representations of each other if for any arbitrary particular context state X the confidence levels of those situations are equal.

Consider an arbitrary context state Y. By definition, the confidence level of situation space *op(X)* can be calculated according to formula (38). Expression (41) provides the confidence for this case.

$$op(Y) = func(Sit_1(Y), \dots, Sit_K(Y)) \quad (41)$$

Context state Y falls into some orthotope of orthotope-based situation space *orthotope(X)* according to lemma 4.1 (see section 4). When assigning the confidence value to that orthotope, we used an arbitrary context state within that orthotope. Let it be state Y'. It means that the confidence level is following (expression (42)).

$$orthotope(Y) = func(Sit_1(Y'), \dots, Sit_K(Y')) \quad (42)$$

Consider any arbitrary situation $Sit_i(X)$ from the list $Sit_1(X) \dots Sit_K(X)$. By definition the situation $Sit_i(X)$ is an orthotope-based situation space, that means according to the lemma 4.1 context state Y belongs to some orthotope of it. Let's refer to that orthotope as: $y_1 \in I(1, i, p_1) \wedge y_2 \in I(2, i, p_2) \dots \wedge y_N \in I(N, i, p_N)$ (the intervals I(a,b,c) are numbered in the same manner as in the expression (39), first index refers to the context attribute, second index refers to the situation in the set, third index refers to particular interval). The same method applies to the context state Y'. It belongs to some orthotope of the orthotope-based situation space $Sit_i(X)$, and let's refer to that orthotope as $y'_1 \in I(1, i, p'_1) \wedge y'_2 \in I(2, i, p'_2) \dots \wedge y'_N \in I(N, i, p'_N)$.

Sub-statement 5.2.3.2: Context states Y and Y' belong to the same orthotope of all the situations $Sit_1(X) \dots Sit_K(X)$. Let's prove it by contradiction. Let the orthotopes for Y and Y' be different for some situation $Sit_i(X)$ from that list. The orthotopes are different, i.e. for at least one context attribute CA_j $y_j \in I(j, i, p_j)$, $y'_j \in I(j, i, p'_j)$ and $p_j \neq p'_j$. By definition of orthotope-based situation space, the intervals $I(j, i, p_j)$ and $I(j, i, p'_j)$ do not overlap.

Now consider the intervals, that situation space *orthotope(X)* has over context attribute CA_j. Consider the formula (40) that shows how those intervals were constructed. Context state attribute y_j belong to some of those intervals, let's refer to it as $I(j, 1, l_1) \cap I(j, 2, l_2) \cap \dots \cap I(j, K, l_K)$. For the context state attribute y'_j let's refer to thatinterval $I(j, 1, l'_1) \cap I(j, 2, l'_2) \cap \dots \cap I(j, K, l'_K)$.

Sub-statement 5.2.3.3. Consider the interval $I(j, 1, l_1) \cap I(j, 2, l_2) \cap \dots \cap I(j, K, l_K)$, described in previous paragraph.

Assertion to prove: the interval (j, i, p_j), defined above, and the interval $I(j, i, l_j)$ is the same interval, i.e. $p_j = l_j$.

Proof. Let's prove it by contradiction. Consider the opposite, i.e. intervals $I(j, i, p_j)$ and $I(j, i, l_j)$ are different. However, the common first index j shows that they are both over the same context attribute, and commons second index *i* shows that they are both from the orthotope-based situation space $Sit_i(X)$. And, by definition of orthotope-based situation

space, within an orthotope-based situation space the intervals over the same context attribute do not overlap. That means, $I(j,i,p_j)$ and $I(j,i,l_j)$ do not overlap, and no particular context attribute value can belong to both of those intervals $I(j,i,p_j)$ and $I(j,i,l_j)$. However, the context attribute value y_j belongs to the intersection $I(j,1,l_1) \cap I(j,2,l_2) \cap \ldots \cap I(j,K,l_K)$, i.e. belongs to every interval in that intersection including $I(j,i,l_j)$. And $y_j \in I(j,i,p_j)$ by definition of that interval. So, y_j belongs to both of the intervals, and therefore they cannot be non-overlapping. It is a contradiction. It means that initial assumption was wrong, and $I(j,i,p_j)$ and $I(j,i,l_j)$ are the same interval.

Q.E.D. for sub-statement 5.2.3.3.

For the same reasons the intervals $I(j,i,p'_j)$ and $I(j,i,l'_j)$ are the same. In order to derive that fact, the proof of sub-statement 5.2.3.3 can be applied to context attribute value y'_j and the intervals $I(j,i,p'_j)$ and $I(j,i,l'_j)$.

Consider the expression (43). It shows the intervals of situation space *orthotope(X)* over context attribute CA_j, that include y_j and y'_j. The intervals $I(j,i,l_j)$ and $I(j,i,l'_j)$, are replaced with the equivalents $I(j,i,p_j)$ and $I(j,i,p'_j)$ respectively.

$$\begin{aligned} y_j \in I(j,1,l_1) \cap I(j,2,l_2) \cap \ldots \cap I(j,i-1,l_{i-1}) \cap I(j,i,p_j) \\ \cap I(j,i+1,l_{i+1}) \cap \ldots \cap I(j,K,l_K) \\ y'_j \in I(j,1,l'_1) \cap I(j,2,l'_2) \cap \ldots \cap I(j,i-1,l'_{i-1}) \cap I(j,i,p'_j) \\ \cap I(j,i+1,l'_{i+1}) \cap \ldots \cap I(j,K,l'_K) \end{aligned} \quad (43)$$

The expression (43) shows that one of the intersections contain $I(j,i,p_j)$, and another intersection contains $I(j,i,p'_j)$. And, as it was already derived, the intervals $I(j,i,p_j)$ and $I(j,i,p'_j)$ do not overlap (see sub-statement 5.2.3.2, paragraph 1).

It means that the intersections from expression (43) do not overlap: there is no particular context attribute value that belongs to both of those intersections, because every particular context attribute value, that belongs to both of those intersections, should belong to every single interval of both of the intersections, including $I(j,i,p_j)$ and $I(j,i,p'_j)$ simultaneously, and there is no particular context attribute value, that belongs to both intervals $I(j,i,p_j)$ and $I(j,i,p'_j)$.

As follow from the previous paragraph, *y* and *y'* belong to non-overlapping intervals of *orthotope(X)* along the context state CA_j. So, the context states Y and Y' cannot be in the single orthotope for situation space *orthotope(X)* – according to formula (22), in order to be in the same orthotope they have to belong to the same intervals for every context attribute, and for at least for CA_j it is not true. However, the context states Y and Y' by their definition (expression (42)) do belong to the same orthotope of *orthotope(X)*. It is a contradiction, and it means that the assumption was wrong, and context state Y and Y' do belong to the same orthotope of any situation $Sit_i(X)$ from the list $Sit_1(X) \ldots Sit_K(X)$. It completes proof by contradiction. **Q.E.D. for sub-statement 5.2.3.2.**

As follows from sub-statement 5.2.3.2, for any situation space $Sit_i(X)$ the context states Y and Y' belong to the same orthotope of the *orthotope(X)*. And according to the formula (22), the confidence level is the same within the orthotope in the situation. Let's refer to it as C_i. It means that $Sit_i(Y') = Sit_i(Y) = C_i$ for any i=1..K.

Summarizing the implications above, the confidence levels of *op(Y)* and *orthotope(Y)* can be rewritten according to the expression (44):

$$op(Y) = func\big(Sit_1(Y), \dots, Sit_K(Y)\big) = func(C_1, C_2, \dots, C_K)$$
$$orthotope(Y) = func\big(Sit_1(Y'), \dots, Sit_K(Y')\big) = func(C_1, C_2, \dots, C_K) \quad (44)$$

According to expression (44), for any arbitrary context state Y the confidence levels of *op(Y)* and *orthotope(Y)* are equal. And by definition it implies that the situation *orthotope(X)* is a representation of *op(X)*.

To summarize, for every arbitrary situation *op(X)*, defined according to the conditions of the lemma 5.2.3, there exists a representation of that situation in the form of orthotope-based situation space. **Q.E.D. for lemma 5.2.3■**

As a result, lemma 5.2.3 implies sufficient conditions for the closure of the orthotope-based situation spaces under a certain operation. Lemma 5.2.4 shows that any arbitrary situation algebra expression is compliant with those sufficient conditions.

Lemma 5.2.4. Premise: Consider an arbitrary situation algebra expression that involves K arbitrary situation spaces. The situation algebra under consideration relies on N basis functions $b_1 \dots b_N$, and those basis functions can be represented as follows (expression (45)).

$$\forall X \epsilon\, \mathbb{St},\ b_1\big(Sit_1, Sit_2, \dots, Sit_{r_1}\big)(X) = \\ = f_1\left(Sit_1(X), \dots, Sit_{r_1}(X)\right);\ f_1\colon \mathbb{C}^{r_1} \to \mathbb{C}$$
$$\forall X \epsilon\, \mathbb{St},\ b_2\big(Sit_1, Sit_2, \dots, Sit_{r_2}\big)(X) = \\ = f_2\left(Sit_1(X), \dots, Sit_{r_2}(X)\right);\ f_2\colon \mathbb{C}^{r_2} \to \mathbb{C} \quad (45)$$
$$\dots$$
$$\forall X \epsilon\, \mathbb{St},\ b_N\big(Sit_1, Sit_2, \dots, Sit_{r_N}\big)(X) = \\ = f_N\left(Sit_1(X), \dots, Sit_{r_N}(X)\right);\ f_N\colon \mathbb{C}^{r_N} \to \mathbb{C}$$

So, according to expression (45) every i-th basis function takes r_i situations as an input, and returns a situation as an output. It should be noted that *situation* (see definition in section 2.2) is merely a function that takes context state as an input and provides a confidence level as an output. For a situation, which can be represented in CST format, the term *situation space* is reserved. The functions f_i, as follows from expressions (45), take r_i confidence levels as an input and provide a confidence level as an output. The functions f_i does not depend on the context state itself.

Lemma statement. Any arbitrary situation algebra expression over orthotope based situation spaces can be represented as an orthotope-based situation space, if for that situation algebra there exists a basis compliant with the definition (45).

Before the proof starts, consider an illustration. Consider the function AND from formula (4). It is compliant with the requirements for a basis function, presented expression (45): by definition for any context state X the confidence of situation *(A & B)(X)* is the minimum of the confidence levels *A(X)* and *B(X)*. It can be formalized as expression (46):

$$\forall X \epsilon \mathbb{St}, (Sit_1\ \&\ Sit_2)(X) = min\big(Sit_1(X), Sit_2(X)\big) \quad (46)$$

For the function AND the count of involved situations is r=2. The function *f* for it is *f(a,b) = min(a,b)* (minimum of confidence levels, or undefined if any of the confidence levels is undefined). Therefore, function AND with the definition presented in formula (45) can be one of the basis functions without breaking the compliance with the definition (45).

Reasoning in the same manner, we can prove that NOT function (formula (4)) is also compliant with the definition (45): for NOT situation algebra function r=1 and *f(a)=1-a* (or undefined if the input is undefined). Taken together, functions AND and NOT constitute

AND-NOT basis, and it means that for the situation algebra definitions presented in formula (4) the basis is compliant with the definition (45), and therefore any situation algebra expression over orthotope-based situation spaces (original CST situation spaces also have orthotope-based representations according to lemma 5.1) is compliant with the conditions of lemma 5.2.4.

Proof. In order to proceed further, we need to prove a sub-statement.

Sub-statement 5.2.4.1. Consider an arbitrary situation algebra expression *Expression(X)*, defined over some situations $Sit_1(X)…Sit_K(X)$. The situations $Sit_1(X)…Sit_K(X)$ can be arbitrary, it is not required for them to be situation spaces of any kind. The basis of situation algebra is compliant with requirement (45).

Assertion to prove (sub-statement 5.2.4.1): For any context state X the expression *Expression(X)* can be viewed as a function of the confidence levels $Sit_1(X)…Sit_K(X)$. It can be formalized as expression (47).

$$\exists func\colon \mathbb{C}^K \to \mathbb{C}\ \text{; s.t. } \forall X \epsilon \mathbb{S}\mathbb{t}, Expression(X) = \\ = func(Sit_1(X), Sit_2(X), \dots, Sit_K(X)) \quad (47)$$

Proof (sub-statement 5.2.4.1). According to the given information the functions $b_1…b_N$ constitute a basis of situation algebra, i.e. any situation algebra expression can be represented as a recursive superposition of those basis functions. We are going to prove the sub-statement by mathematical induction over the recursion depth. This depth will be referred to as *n*.

Induction basis. The depth *n=0* means a degenerate situation algebra expression that consists of just an input situation itself, and does not require any basis functions: $Expression(X)=Sit_1(X)$. Therefore, $func\big(Sit_1(X)\big) = Sit_1(X)$fits the requirements (47), and the fact that we found the function proves its existence. **Q.E.D. for induction basis.**

Induction step. Consider that the functions exist for the situation algebra expressions that can be calculated by applying basis function recursively, if the recursion depth is not more than *n-1*. Let's prove that the function exists for the expressions that require *n* depth of recursion.

Consider an arbitrary situation algebra expression *Expression(X)*, defined over situations $Sit_1(X)…Sit_K(X)$, and the situation *Expression(X)* can be calculated using basis functions recursively with the depth not more than *n*. During the calculations, some basis function from the set $b_1…b_N$ is going to be calculated last, let's refer to that function as b_i. In this case, using the conditions (45), the expression *Expression(X)* can be represented as follows (formula (48)).

$$\forall X \epsilon \mathbb{S}\mathbb{t}, Expression(X) = b_i\big(G_1, G_2, \dots, G_{r_i}\big)(X) = (G_1(X), G_2(X), \dots, G_{r_i}(X)) \quad (48)$$

The sub-expressions $G_j(X)$ are later going to be calculated recursively using the basis functions.

The depth of recursion for expression *Expression(X)* exceeds by one the maximum depth of recursion, required to calculate any $G_j(X)$ (calculation of b_i on top of entire $G_j(X)$ recursive calculations). And by definition the depth of *Expression(X)* is not greater than *n*. It means that for the situations $G_j(X)$ recursion depth is not more than *n-1* (if *n-1=0*, it means that $G_j(X)$ are just the input situations). Therefore, according to induction assumption, for any situation $G_j(X)$ there exist a function, compliant with properties (47). Let's refer to it as $func_j(Sit_1(X), Sit_2(X),…,Sit_K(X))$. Without the loss of generality, we can consider that $func_j(…)$ is defined over all the situation spaces $Sit_1(X), Sit_2(X),…,Sit_K(X)$ (if

any of those do not influence the *func$_j$(…)* output, it does not break the proof). Therefore, the expression (48) can be rewritten as expression (49).

$$\forall X \epsilon \mathbb{St}, Expression(X) = b_i\left(G_1, G_2, \dots, G_{r_2}\right)(X) = \\ = f_i(func_1(Sit_1(X), Sit_2(X), \dots, Sit_K(X)), \\ \dots, func_{r_i}(Sit_1(X), Sit_2(X), \dots, Sit_K(X)) \quad (49)$$

Consider a definition of function *func*$(c_1, c_2, \dots, c_K)$, that takes K confidence levels as an input and produces confidence level as an output (expression (50)).

$$func\colon \mathbb{C}^K \rightarrow \mathbb{C}\ ;\ func(c_1, c_2, \dots, c_K) \equiv \\ \equiv f_i(func_1(c_1, c_2, \dots, c_K), \dots, func_{r_i}(c_1, c_2, \dots, c_K)) \quad (50)$$

Using the definition (50), the expression (49) can be rewritten as formula (51):

$$\forall X \epsilon \mathbb{St}, Expression(X) = func(Sit_1(X), Sit_2(X), \dots, Sit_K(X)), \quad (51)$$
$$\text{where } func\colon \mathbb{C}^K \rightarrow \mathbb{C}$$

Expression (51) directly shows that the function *func*$(c_1, c_2, \dots, c_K)$ fits the requirements (47). And the fact that we found the function proves its existence.

Q.E.D. for induction step, and that completes the proof of sub-statement 5.2.4.1.

Now let's return to the proof of the main part of lemma 5.2.4.

Consider an arbitrary situation algebra expression *Expression(X)*, defined over some orthotope-based situations *Sit$_1$(X)…Sit$_K$(X)*. The basis of situation algebra is compliant with the properties (45).

Consider several already proven statements:

1. By definition there exists a set of K orthotope-based situation spaces *Sit$_1$(X)…Sit$_K$(X)*.
2. According to the sub-statement 5.2.4.1, there exists a function $func\colon \mathbb{C}^K \rightarrow \mathbb{C}$, so that $\forall X \epsilon \mathbb{St}, Expression(X) = func(Sit_1(X), Sit_2(X), \dots, Sit_K(X))$

The statements 1 and 2, taken together, constitute the conditions of lemma 5.2.3, where $Expression(X)$ stands for *op(X)*. Therefore, according to lemma 5.2.3 the situation $Expression(X)$ has an orthotope-based representation. **Q.E.D. for lemma 5.2.4■**

Lemma 5.2.4 shows that the orthotope-based situation space is closed under any situation algebra expression, if the basis functions of the situation algebra are compliant with expression (45). For example, all the situation algebra methods from formula (4) are compliant with those conditions. See expressions (52).

AND: (Sit$_1$&Sit$_2$(X)) = f(Sit$_1$(X), Sit$_2$(X)),
where *f(a,b) = min(a,b)* (or *UD* if either *a* or *b* is *UD*).
OR: (Sit$_1$|Sit$_2$)(X) = f(Sit$_1$(X), Sit$_2$(X)), (52)
where *f(a,b) = max(a,b)* (or *UD* if either *a* or *b* is *UD*).
NOT: (¬*Sit)(X) = f(Sit(X))*, where *f(a) = 1-a* (or *UD* if *a* is *UD*).

Lemma 5.2.4 also implies the closure, if instead of Zadeh operators [Za65] from formula (4) the user chooses the functions like expressions (53), or expressions (54).

AND: $(Sit_1 \& Sit_2(X)) = f(Sit_1(X), Sit_2(X)))$,
where $f(a,b) = min(0;a+b-1)$ (or *UD* if either *a* or *b* is *UD*).
OR: $(Sit_1 | Sit_2)(X) = f(Sit_1(X), Sit_2(X)))$,). (53)
where $f(a,b) = max(1;a+b)$ (or *UD* if either *a* or *b* is *UD*).
NOT: $(\neg Sit)(X) = f(Sit(X))$, where $f(a) = 1-a$ (or *UD* if *a* is *UD*).

AND: $(Sit_1 \& Sit_2(X)) = f(Sit_1(X), Sit_2(X)))$,
where $f(a,b) = a*b$ (or *UD* if either *a* or *b* is *UD*).
OR: $(Sit_1 | Sit_2)(X) = f(Sit_1(X), Sit_2(X)))$, (54)
where $f(a,b) = a+b-a*b$ (or *UD* if either *a* or *b* is *UD*).
NOT: $(\neg Sit)(X) = f(Sit(X))$, where $f(a) = 1-a$ (or *UD* if *a* is *UD*).

Lemma 5.2.4 ensures the existence of orthotope-based situation representation. However, in order to derive the verification result, that orthotope-based situation representation needs to undergo an emptiness check. And in order to do this, the orthotope-based situation representation needs to be derived explicitly. Consider the algorithm 5.2 that originates in the proofs of lemma 5.2.3.

Algorithm 5.2. Input. A set of orthotope-based input situations $Sit_1(X) \ldots Sit_K(X)$ and a situation *Expression(X)*, that complies with the property (55).

$$\forall X \epsilon \mathbb{S}\mathbb{t}, Expression(X) = func(Sit_1(X), Sit_2(X), \ldots, Sit_K(X)) \quad (55)$$

It should be specially noted that the knowledge of function *func(…)* is not required, the condition is just that it exists. The situation *Expression(X)* can as well be a blackbox. For example, the situation *Expression(X)* can be a situation algebra expression, calculated directly using the superposition of formulas (4).

Expected output: orthotope-based representation of *Expression(X)*.

Algorithm steps. For better understandability, the algorithm is presented as a set of steps.

Step 1. Construct a set of involved context attributes $CA_1 \ldots CA_N$. The context attribute is added to the set, of it is involved in at least one of the situations $Sit_1(X) \ldots Sit_K(X)$.

Step 2. For every situation $Sit_i(X)$ within $Sit_1(X) \ldots Sit_K(X)$ and for every context attribute CA_j within $CA_1 \ldots CA_N$ do step 2.1.

Step 2.1. If the situation $Sit_i(X)$ does not involve context attribute CA_j, add it into consideration in compliance with lemma 5.2.1. For numeric context attribute the interval $x_j \in (-\infty,+\infty)$ can be added to the description of every orthotope. For non-numeric context attribute the interval $x_j \in \neg\{\}$ (the list of non-included non-numeric values is empty, i.e. all non-numeric values are included) can be added to the description of every orthotope, or it can be the interval that contains all possible non-numeric values explicitly. The number of orthotopes will not be increased in those cases. For mixed context attributes, those methods should be combined, and thus the number of the orthotopes will double. Let's refer to the count of intervals that the situation $Sit_i(X)$ has over context state CA_j as r(i,j) and to the intervals themselves as I(i,j,1)… I(i,j,r(i,j)).

Step 3. Create an orthotope-based situation space *orthotope(X)* over the context attributes $CA_1 \ldots CA_N$.

Step 4. For every context attribute CA_j within $CA_1 \ldots CA_N$ do step 4.1.

Step 4.1. For any combination of intervals $I(i,1,p_1)$, $I(i,2,p_2)$, …, $I(i,K,p_k)$, where $p_1 = 1..r(i,1)$, $p_2 = 1..r(i,2)$, …, $p_K = 1..r(i,K)$ do the steps 4.1.1.-4.1.2.

Step 4.1.1. Calculate an interval $I(i,1,p_1) \cap I(i,2,p_2) \cap \ldots \cap I(i,K,p_k)$.

Step 4.1.2. If it is not empty, add it to *orthotope(X)* as a part of

decomposition of CA_j.

Step 5. For every orthotope of *orthotope(X)* do steps 5.1.- 5.3.

Step 5.1. Generate a random context state X' within the considered orthotope.

Step 5.2. Calculate the confidence level *Expression(X')*.

Step 5.3. Assign confidence value *Expression(X')* to the orthotope under consideration.

Output. Situation *orthotope(X)*

Correctness Proof. The correctness of the algorithm is implied by lemma 5.2.3. The algorithm 5.2 represents in details the steps of lemma 5.2.3 to construct the orthotope-based representation *orthotope(X)* out of the situation space *op(X)* (here – *Expression(X)*). And the equivalence of *orthotope(X)* and *op(X)* is proven as a part of lemma 5.2.3. **Q.E.D.** ■

The complexity of the algorithm 5.2 is evaluated in section 6. Consider an illustration of the algorithm 5.2. Refer to the sample scenario, presented in section 3.2. In that scenario we need to verify whether the situation *LightMalfunctions(X)* (definition (20)) and *ConditionsAcceptable(X)* (definition (19)) are in a contradiction. The contradiction verification implies checking that the expression (*LightMalfunctions & ConditionsAcceptable)(X)* is empty.

Having the orthotope-based representations for the involved situations (formulas (23) and (28)), let's derive the situation (*LightMalfunctions & ConditionsAcceptable)(X)* in an orthotope-based format, using the algorithm 5.2. Each step of the algorithm is addressed briefly.

Step 1. The involved context attributes are CA_1=*LightLevel*, CA_2= *NoiseLevel* and CA_3=*SwitchPosition*.

Step 2. Adding the interval *NoiseLevel*$\in$ (-∞,+∞) to *LightMalfunctions(X)* and *SwitchPosition* $\in$ *{On,Off}* to *ConditionsAcceptable(X)*. The results (in a concise manner) are presented in the expression (56).

$$ConditionsAcceptable(X) =$$
$$= \begin{bmatrix} 0.5, (LightLevel < 350) \wedge (NoiseLevel \leq 40) \wedge (SwitchPosition \in \{On, Off\}) \\ 0.35, (LightLevel < 350) \wedge (NoiseLevel \in [40{,}50)) \wedge (SwitchPosition \in \{On, Off\}) \\ \dots \\ 0.5, (LightLevel \geq 500) \wedge (NoiseLevel > 60) \wedge (SwitchPosition \in \{On, Off\}) \end{bmatrix}$$

$$LightMalfunctions(X) = \quad (56)$$
$$= \begin{bmatrix} 0.5, (LightLevel<350) \wedge (SwitchPosition \in \{Off\}) \wedge (NoiseLevel \in (-\infty, +\infty)) \\ 0.25, (LightLevel \in [350{,}500)) \wedge (SwitchPosition \in \{Off\}) \wedge (NoiseLevel \in (-\infty, +\infty)) \\ \dots \\ 0.5, (LightLevel \geq 500) \wedge (SwitchPosition \in \{On\}) \wedge (NoiseLevel \in (-\infty, +\infty)) \end{bmatrix}$$

Step 3. Creating *orthotope(X)* over context attributes are CA_1=*LightLevel*, CA_2=*NoiseLevel* and CA_3=*SwitchPosition*.

Step 4. The possible combinations of intervals over the context attributes are following (expression (57), the empty intervals are already removed):

LightLevel: I(1,1) – LightLevel $<$ 350; *I(1,2) – LightLevel* $\in$ *[350,500); I(1,3) – LightLevel*$\geq$*500*
NoiseLevel: I(2,1) – NoiseLevel $\leq$ 40; *I(2,2) – NoiseLevel* $\in$ *(40,50];* (57)
I(2,3) – NoiseLevel $\in$ *(50,60]; I(2,4) – NoiseLevel*>*60*
SwitchPosition: I(3,1) – SwitchPosition $\in$ {*On*}; *I(3,2) - SwitchPosition* $\in$ {Off}

Therefore, the context attributes of *orthotope(X)* are decomposed into intervals, presented in expression (57). The situation *orthotope(X)* will consist of 24 orthotopes – every orthotope will correspond to a combination of one interval over *LightLevel*, one interval over *NoiseLevel* and one interval over SwitchPosition..

Step 5. Let's illustrate the step 5 on one of the orthotopes. The remaining orthotopes are processed in a similar manner. Consider the orthotope: $LightLevel \in [350,500)$; $NoiseLevel \leq 40$; $SwitchPosition \in \{Off\}$.

Step 5.1. Generate random context state inside the orthotope. Let it be: X' = $\{LightLevel = 400; NoiseLevel = 30; SwitchPosition = \{Off\}\}$.

Step 5.2. Calculate the confidence value of the expression at the generated context state X'. The calculation process is presented in the expression (58).

$$\begin{aligned}&(LightMalfunctions \ \&\ ConditionsAcceptable)(X') = \\ &= min(LightMalfunctions(X'); ConditionsAcceptable(X')) = \\ &= min(0.25; 0.75) = 0.25\end{aligned} \quad (58)$$

Step 5.3. Assign the obtained confidence value to the orthotope under consideration. It means, one of the rows of *orthotope(X)* definition will be following (expression (59)).

$$0.25;\ LightLevel \in [350,500);\ NoiseLevel \leq 40;\ SwitchPosition \in \{Off\} \quad (59)$$

The orthotope-based representation of (*LightMalfunctions & ConditionsAcceptable)(X)* emerges as the results of the algorithm 5.2. We will need that situation to illustrate emptiness check, so we present the complete result of the algorithm in the expression (60).

$$(LightMalfunctions \ \&ConditionsAcceptable)(X) = \begin{bmatrix} 0.5, (LightLevel < 350) \wedge (NoiseLevel \leq 40) \wedge (SwitchPosition \in \{On\}) \\ 0.35, (LightLevel < 350) \wedge (NoiseLevel \in [40,50)) \wedge (SwitchPosition \in \{On\}) \\ 0.15, (LightLevel < 350) \wedge (NoiseLevel \in [50,60)) \wedge (SwitchPosition \in \{On\}) \\ 0, (LightLevel < 350) \wedge (NoiseLevel > 60) \wedge (SwitchPosition \in \{On\}) \\ 0.75, (LightLevel \in [350,500)) \wedge (NoiseLevel \leq 40) \wedge (SwitchPosition \in \{On\}) \\ 0.6, (LightLevel \in [350,500)) \wedge (NoiseLevel \in [40,50)) \wedge (SwitchPosition \in \{On\}) \\ 0.4, (LightLevel \in [350,500)) \wedge (NoiseLevel \in [50,60)) \wedge (SwitchPosition \in \{On\}) \\ 0.25, (LightLevel \in [350,500)) \wedge (NoiseLevel > 60) \wedge (SwitchPosition \in \{On\}) \\ 0.5, (LightLevel \geq 500) \wedge (NoiseLevel \leq 40) \wedge (SwitchPosition \in \{On\}) \\ 0.5, (LightLevel \geq 500) \wedge (NoiseLevel \in [40,50)) \wedge (SwitchPosition \in \{On\}) \\ 0.5, (LightLevel \geq 500) \wedge (NoiseLevel \in [50,60)) \wedge (SwitchPosition \in \{On\}) \\ 0.5, (LightLevel \geq 500) \wedge (NoiseLevel > 60) \wedge (SwitchPosition \in \{On\}) \\ 0.5, (LightLevel < 350) \wedge (NoiseLevel \leq 40) \wedge (SwitchPosition \in \{Off\}) \\ 0.35, (LightLevel < 350) \wedge (NoiseLevel \in [40,50)) \wedge (SwitchPosition \in \{Off\}) \\ 0.15, (LightLevel < 350) \wedge (NoiseLevel \in [50,60)) \wedge (SwitchPosition \in \{Off\}) \\ 0, (LightLevel < 350) \wedge (NoiseLevel > 60) \wedge (SwitchPosition \in \{Off\}) \\ 0.25, (LightLevel \in [350,500)) \wedge (NoiseLevel \leq 40) \wedge (SwitchPosition \in \{Off\}) \\ 0.25, (LightLevel \in [350,500)) \wedge (NoiseLevel \in [40,50)) \wedge (SwitchPosition \in \{Off\}) \\ 0.25, (LightLevel \in [350,500)) \wedge (NoiseLevel \in [50,60)) \wedge (SwitchPosition \in \{Off\}) \\ 0.25, (LightLevel \in [350,500)) \wedge (NoiseLevel > 60) \wedge (SwitchPosition \in \{Off\}) \\ 0, (LightLevel \geq 500) \wedge (NoiseLevel \leq 40) \wedge (SwitchPosition \in \{Off\}) \\ 0, (LightLevel \geq 500) \wedge (NoiseLevel \in [40,50)) \wedge (SwitchPosition \in \{Off\}) \\ 0, (LightLevel \geq 500) \wedge (NoiseLevel \in [50,60)) \wedge (SwitchPosition \in \{Off\}) \\ 0, (LightLevel \geq 500) \wedge (NoiseLevel > 60) \wedge (SwitchPosition \in \{Off\}) \end{bmatrix} \quad (60)$$

5.3 Emptiness Check for an Orthotope-based Situation Space

As proposed in section 3.1, the verification of situation relationship can be viewed as an emptiness check of a certain situation algebra expression. Any situation algebra expression over original CST situation spaces can be represented as an orthotope-based situation space

in two steps. At first, the involved situations should be represented in an orthotope-based situation space format using the algorithm 5.1. Then the orthotope-based representation of the situation algebra expression can be derived using the algorithm 5.2. After that, when the orthotope-based representation of situation algebra expression is obtained, it requires only emptiness check of that situation space in order to complete the verification.

Emptiness of the orthotope-based situation space can be checked using the algorithm 5.3.

Algorithm 5.3. Input.

1. Orthotope-based situation space *S(X)*, that is defined over the context attributes $CA_1 \ldots CA_N$ according to the definition (22).
2. Threshold *th*. The emptiness is checked with respect to that threshold (see (9)).
3. The flag *searchAllCounterexamples*. The algorithm can either perform an extensive search and find all possible counterexamples, or just derive yes/no answer.

Expected output: The set of counterexamples, i.e. the definitions of context states, where the situation is occurring. All possible counterexample should be in the counterexample set.

Algorithm pseudocode.

```
CounterexampleSet cs = new CounterexampleSet(); //Initializing the empty set
for every Ci from  S.orthotopes() //For every orthotope, L orthotopes in total
   //If the orthotope has the confidence over the threshold
   L1: if (Ci.confidence() >= th) then
      //Add the definition of orthotope to the counterexamples
      cs.add(Ci.orthotopeDescription());
      //Return cs, if only yes/no answer is needed.
      L2: if (¬searchAllCounterexamples)  then return cs;
  end if
end for
return cs;
```

Output. Counterexample set *cs*, that contains the definitions of all the orthotopes, which have the confidence value over the threshold.

Correctness proof.

The correctness proof consists of 2 asserted statements:

Statement 1. The algorithm cannot take a wrong value in the counterexample set.

Statement 2. The algorithm cannot skip the counterexample, if *searchAllCounterexamples* is on.

Proof of statement 1. Let's prove it by contradiction. Consider that there exists a context state X, for which the confidence value *S(X)* is lower than threshold, but the value itself does belong to the counterexample set.

According to lemma 4.1 the context state X belongs to some single orthotope. Let's refer to it as the orthotope C_i. By definition of the algorithm, the counterexample set is the set of orthotopes of *S(X)*. It means, that in order for context state X to belong to counterexample set, the whole orthotope that contains it should belong to counterexample set. And the only set that contains X is the set C_i, so it should be in the counterexample set.

According to formula (22) the confidence level at context state X is equal to the confidence level associated with the orthotope C_i. By definition of the algorithm, if the orthotope confidence is lower than threshold, it is not added to the counterexample set: it

will either fail the condition at label (L1), or the return will be triggered earlier at label (L2). But in either case the orthotope C_i should not have been added to the counterexample set.

As a summary, the orthotope C_i should and should not be in the counterexample set at the same time. It is a contradiction. So, initial assumption was wrong, and there is no context state X, that belongs to the counterexample set, but has the confidence value *S(X)* below the threshold. **Q.E.D. for statement 1.**

Proof of statement 2. Statement 2 applies only to the case when *searchAllCounterexamples=true*, so the condition at label (L2) is always false. Let's take an arbitrary counterexample and prove that it will belong to the counterexample set. Consider an arbitrary context state X, for which the *S(X)* reaches the threshold: $S(X) \geq th$. According to lemma 4.1 the context state X belongs to some orthotope of the *S(X)*. Let's refer to that orthotope as C_i. According to the definition (22), the confidence level at state X is the confidence level, associated with the corresponding orthotope. It implies that $S(X) = C_i.confidence$, which in turn means that $C_i.confidence \geq th$.

However, if *searchAllCounterexamples=true* than the algorithm will parse through all the orthotopes. Without triggering the condition at label (L2), the loop cannot end before iterating through all the orthotopes. Consider the iteration of the loop, that concerns the orthotope C_i. As it was already proven, $C_i.confidence \geq th$. It means, condition at label (L1) will be triggered, and as a result the whole orthotope will be added to the counterexample set.

As a result, the orthotope C_i is a sub-state of the counterexample set. The context state X belongs to the sub-state C_i of the counterexample set, and therefore to the counterexample set as well.

So, if *searchAllCounterexamples* is on, than any arbitrary context state X, for which $S(X) \geq th$, will belong to the counterexample set. **Q.E.D. for statement 2. ■**

Consider an example of the algorithm 5.3 applied to the example scenario from section 3.2. The situation of *LightMalfunctions(X)* should be contradictory with *ConditionsAcceptable(X)*, and therefore the situation of (*LightMalfunctions & ConditionsAcceptable)(X)* should be empty w.r.t. to the chosen threshold 0.7. Using the algorithm 5.2, the situation (*LightMalfunctions & ConditionsAcceptable)(X)* was represented as an orthotope-based situation space in expression (60).

The algorithm 5.3 iterates the situation orthotope after orthotope (i.e. row after row in formula (22)), and adds orthotope description to the counterexample set, if the orthotope confidence level reaches the threshold. As follows from formula (60), the only orthotope where the confidence level reaches the threshold 0.7 is the orthotope $LightLevel \in [350{,}500) \wedge NoiseLevel \leq 40 \wedge SwitchPosition \in \{On\}$. It is a counterexample.

The verification of the example scenario is complete. The results of the verification process are following:

1. The specification does not comply with the expected relationship. The situations *LightMalfunctions(X)* and *ConditionsAcceptable(X)* do not have contradiction relationship over the entire application space.
2. For any particular context state within the orthotope $LightLevel \in [350{,}500) \wedge NoiseLevel \leq 40 \wedge SwitchPosition \in \{On\}$ the contradiction relationship between *LightMalfunctions(X)* and *ConditionsAcceptable(X)* will be broken.
3. For any other context state, the contradiction relationship will hold true.

In the next section we are going to summarize the approach and propose the method for the verification of an arbitrary situation relationship.

5.4 Verification of Situation Specifications

According to the set of algorithms and lemmas, provided in the sections 5.1-5.3, the formal verification of an arbitrary situation relation can be performed as follows:

1. Represent the property under verification as a situation algebra expression that should be checked for emptiness. The guidelines for deriving the representations are given in section 3.
2. Convert the involved situations to orthotope-based situation spaces using the algorithm 5.1.
3. Using the situation algebra expression under test and the converted input situations, derive the orthotope-based representation of the expression using the algorithm 5.2.
4. Check orthotope-based representation for emptiness using the algorithm 5.3 and obtain the counterexamples, if needed.

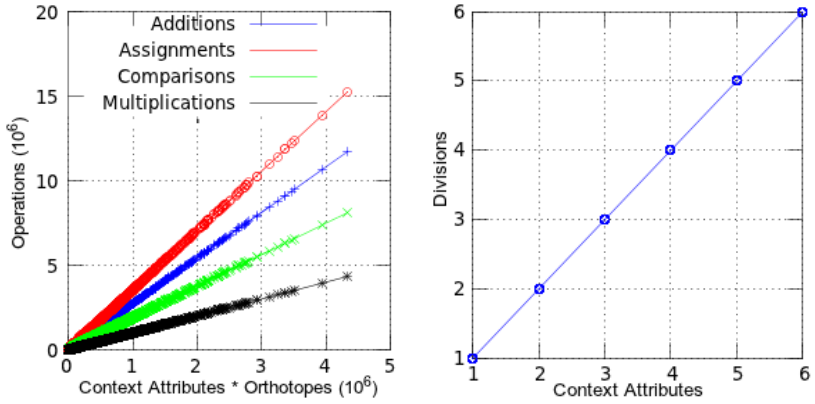
The theoretical complexity analysis and the evaluation of the verification approach is provided in section 6.

6 Formal Verification Mechanism Evaluation and Complexity Analysis

The proposed formal verification mechanism consists of three algorithms 5.1-5.3, respectively the conversion of situation format, retrieving the orthotope-based representation of the expression and emptiness check. Those algorithms will be analyzed in detail in sections 6.1-6.3. The summary of verification evaluation and complexity analysis will be provided in the section 6.4.

6.1 The Conversion of Situation Format

The first step of the verification process is the conversion of the involved situations into the orthotope-based format using the algorithm 5.1. Theoretical and practical evaluation of the algorithm 5.1 is discussed below. For evaluation we used ECSTRA (Enhanced Context Spaces Theory Reasoning Architecture) – a context spaces theory based framework for context awareness and situation awareness. The verification mechanisms were implemented as an extension of ECSTRA. Due to the space requirements the detailed description of ECSTRA framework is omitted, an interested reader can refer to [BZ11a][BZ11c]. The experiment was carried out as follows: 50000 original CST situations were generated randomly using the specially developed ECSTRA add-on. For each situation the number of involved context attributes was generated uniformly then each context attribute was decomposed into the set of intervals, which were also generated uniformly. Using the algorithm 5.1, implemented as a part of ECSTRA verification mechanism, each of the situations was converted into the orthotope-based format. The practical complexity in terms of different operations was calculated using the counters. The experiments were conducted on Lenovo ThinkPad T61 (2.50GHz processor, 2GB RAM, Ubuntu Linux OS). We used R [VS12] and Octave [Ea11] applications for data processing and generating the plots. The experimental results are presented in the figure 2 and analyzed in table 1.



(a) Additions, Multiplications, Comparisons, Assignments (b) Divisions

Fig. 2. The complexity of the algorithm 5.1

Table 1 contains the explanation and theoretical analysis of the results, presented in figure 2. For every involved operation the table 1 contains the order of the operation, practical R^2 and theoretical evaluations. The coefficient R^2 is calculated from the experimental data using R [VS12] statistical package. The data used for constructing figure 2 and for calculating R^2 are the same. R^2 coefficient practically proves the theoretically expected order of operations by showing the fit between the expected order and the number of operations of certain kind. In compliance with the definition of orthotope-based situation space (section 4), N stands for the number of involved context attributes and L stands for the number of orthotopes in resulting orthotope-based situation space.

To summarize, the complexity of the algorithm is O(N*L) – the order of count of orthotopes multiplied by the number of involved context attributes.

6.2 Orthotope-based Representation of Expression

As the section 6.1 demonstrates, the involved situations can each be converted to the orthotope representation at the order of O(N*L), where N is the number of involved context attributes, and L is the number of orthotopes. After all the situations are converted into the orthotope-based situation spaces, the resulting situation can be derived using the algorithm 5.2.

The algorithm 5.2 takes as an input all the involved situations and the situation $Expression(X) = func(Sit_1(X), Sit_2(X), \ldots, Sit_K(X))$, that can as well be a blackbox. For evaluation purposes we estimate the complexity for the operations AND, OR and NOT, defined by formulas (4). The experiment was carried out in a similar manner as in the section 6.1. The algorithm 5.2 was implemented as a part of ECSTRA verification extension, and the random situation generation was done using the specially designed add-on. CST situation spaces were randomly generated in a manner, similar to the experiment in

Table 1. The Complexity of the Algorithm 5.1

Operation	Order	R^2	Explanation
Additions	O(N*L)	0.9999	The resulting situation space contains L orthotopes. For every orthotope the calculation of the confidence value contains a sum with N summands. Also the algorithm contains numerous auxiliary additions, related to various loops.
Multiplications	O(N*L)	1	The resulting situation contains L orthotopes and in order to determine the confidence level of each orthotope, a sum with N summands should be calculated. Every summand involves one multiplication, so the complexity is O(N*L) for multiplications.
Divisions	O(N)	1	The division operations is used to normalize the weights of the basic situation spaces (make them sum up to 1) before the conversion starts. There are N weights, and each of the weights is divided by the sum of other weights.
Assignments	O(N*L)	0.9999	The main portion of the assignments appears as the part the orthotope description creation. For each of the L orthotopes the description is composed of N intervals, and that results in O(N*L) order.
Comparisons	O(N*L)	0.9997	Although the comparisons are not explicitly present in the algorithm 5.1, they are involved in the loops. Each iteration of the loop contains a comparison in order to test for the exit condition. The most comparison-heavy part, which explains the order, is the calculation of the confidence levels. For each of the L orthotopes there is a nested loop, which calculates the weighted sum of N contributions. It makes the number of comparisons O(N*L).

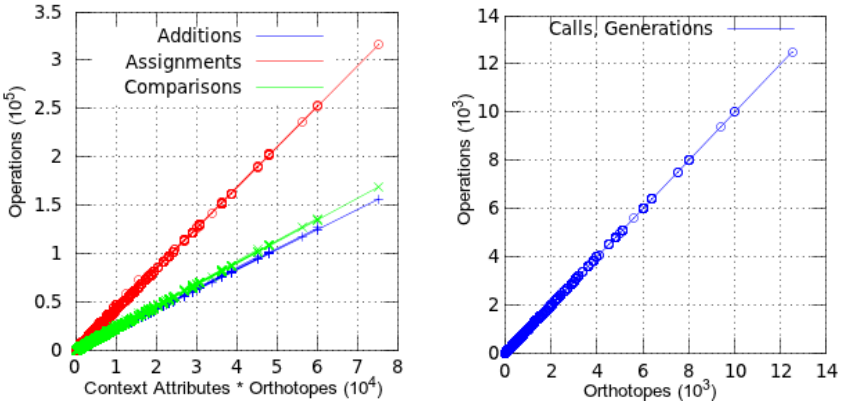
section 6.1 (50 000 situations for NOT, 10 000 situation pairs for AND and OR). The conversion of the situations into orthotope-based situation spaces was evaluated in section 6.1, and therefore the operations for conversion are not counted in this experiment. The converted situations (or a single situation in case of NOT) were used as an input for the algorithm 5.2. The results of the experiment are presented in the figures 3 (AND), figure 4 (OR) and figure 5 (NOT). The theoretical complexity calculations and their connection to the practical results are discussed in table 2, which is designed similarly to table 1. There are three coefficients R^2 provided – one for each of for the experiments (for AND, OR and NOT respectively).

To summarize, the theoretical complexity of the representation algorithm is O(N*L) and, according to the figures 3-5 and R^2 coefficient in table 2, the practical evaluation is absolutely compliant with the theoretical results.

It might seem counterintuitive that there is no dependency on the number of involved situations (let's refer to that number as K, in compliance with the section 5.2). However, that number is implicitly contained in various characteristics and heavily influences the total complexity of the algorithm. The number of the involved situations K and the structure of those situations do influence the decomposition of context attributes into resulting intervals. In turn, that decomposition defines the number of resulting orthotopes, and therefore has a determining influence on the total complexity of the algorithm. The number of situations also can influence the complexity of the expression call. The influence of number of involved situations will also be addressed as the part of the final complexity evaluation in section 6.4.

Table 2. The Complexity of the Algorithm 5.2

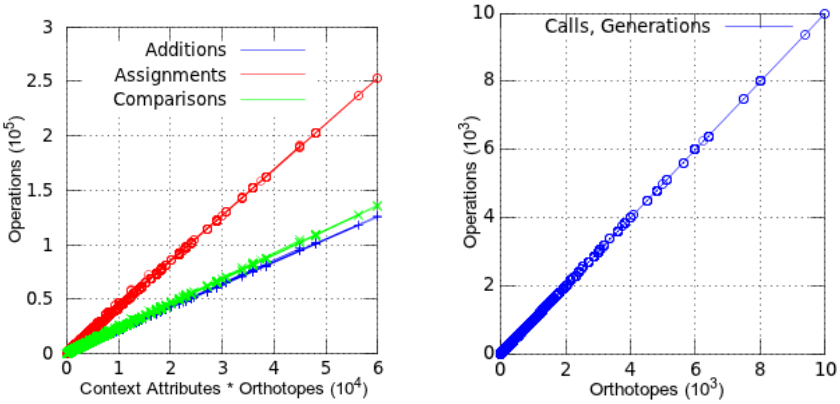
Operation	Order	R^2	Explanation
Additions	O(N*L)	AND: 0.9994 OR: 0.9994 NOT: 0.9979	The main sources of additions are the nested loops, which use additions to increase the counters. Calculation of confidence levels involves context state generations. To generate the context state for each of the L orthotopes N context attribute values need to be generated. The nested loop gives N*L additions for the counters, and that results in O(N*L) order.
Assignments	O(N*L)	AND: 0.9995 OR: 0.9995 NOT: 0.9987	As well as for the addition operation, the part of the algorithm, which determines the order of assignments, is generation of context state. The nested loop for context state generation adds O(N*L) assignments to the total algorithm complexity.
Comparisons	O(N*L)	AND: 0.9994 OR: 0.9994 NOT: 0.9982	The most comparison-heavy part of the algorithm is the nested loop for generating the test context state. It results in the order O(N*L).
Expression calls	O(L)	1 (in all cases)	In the algorithm 5.2 the confidence level of the situation op(*X*) is calculated exactly once for every orthotope, in order to determine the confidence value.
Random value generation	O(N*L)	1 (in all cases)	Random value generation is performed N times for each of the L orthotopes, in order to generate N-valued context state and use it as an input for the expression calculation.



(a) Additions, assignments, comparisons

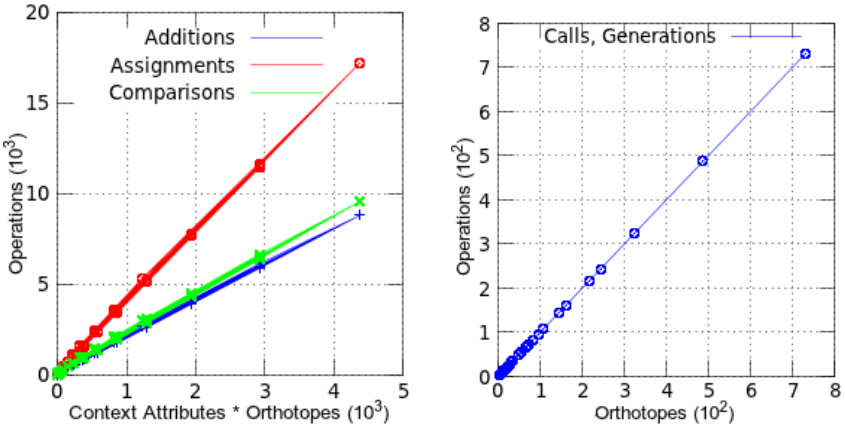
(b) Expression calls, context state generation

Fig. 3. The complexity of the algorithm 5.2 for AND operation.



(a) Additions, assignments, comparisons (b) Expression calls, context state generation

Fig. 4. The complexity of the algorithm 5.2 for OR operation.



(a) Additions, assignments, comparisons (b) Expression calls, context state generation

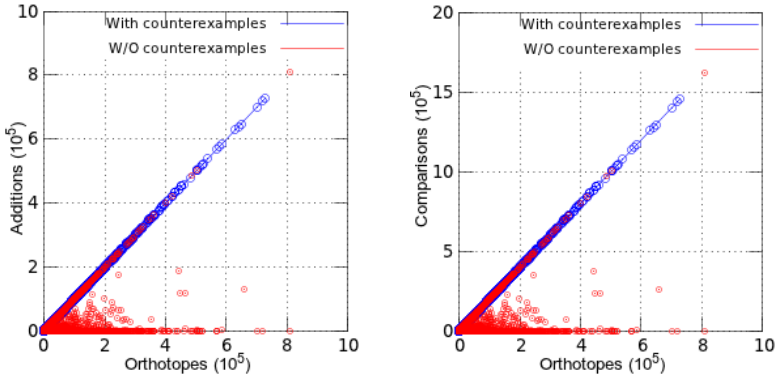
Fig. 5. The complexity of the algorithm 5.2 for NOT operation.

6.3 Emptiness Check

After the orthotope-based representation of the situation space is produced using the algorithm 5.2, it should be checked for emptiness using the algorithm 5.3. The experiment was carried out in a similar way as the experiment for conversion test in the section 6.1. The algorithm 5.3 was implemented as a part of ECSTRA verification extension, and the random situation generation was performed using the specially designed add-on. The results of the evaluation of the algorithm 5.3 are presented in the table 3 and the figure 6. In the figure 6 blue marks identify the case when all the counterexamples had to be explicitly

computed (*searchAllCounterexamples=true* in the algorithm 5.3). The red marks of the figure 6 represent the case when the algorithm stops after finding a single counterexample (*searchAllCounterexamples=false* in algorithm 5.3). It should be noted that the cases *searchAllCounterexamples=true* and *searchAllCounterexamples=false* had different sets of generated situations. As expected, the practical complexity in the case *searchAllCounterexamples=false* varies between a single loop iteration as the lower bound and the complexity of *searchAllCounterexamples=true* as the upper bound. The table 3 refers both to the case *searchAllCounterexamples=true* and to the upper bound of the case *searchAllCounterexamples=false*.

The analysis of emptiness check algorithm, presented in table 3, shows O(L) complexity. This estimation is true for both the case *searchAllCounterexamples=true*, and the upper border for the case *searchAllCounterexamples=false*. This result finalizes the complexity evaluation of the algorithms 5.1-5.3 of the verification process. Section 6.4 summarizes the verification mechanism evaluation and discusses the derived total complexity of the verification process.



(a) Additions (b) Comparisons

Fig. 6. The complexity of the algorithm 5.3.

Table 3. The Complexity of the Algorithm 5.3

Operation	Order	R^2	Explanation
Comparisons	O(L)	1	In the algorithm 5.3 The main loop contains L iterations. In every iteration of the main loop the confidence level is tested against the threshold, and also the loop condition is tested against the exit condition. It makes the number of comparison operations at the order of O(L).
Additions	O(L)	1	Additions are used within the main loop, in order to increase the loop counter, once per iteration.
Assignments	O(L)	1	The assignments are used as the part of the main loop in order to increase the loop counter.

6.4 Verification of Situation Definitions – Total Complexity

The complexity of the formal verification mechanism aggregates the complexity of the algorithms 5.1-5.3, which is evaluated in the sections 6.1-6.3. Combination of the results, provided in sections 6.1-6.3, allows us to derive the following complexity of the verification process (formula (61)).

$$O(\sum_{i=1}^{K} N_i L_i + N_{res} L_{res} + L_{res}) \quad (61)$$

In formula (61) N_i and L_i (i=1..K) stand for the number of orthotopes and number of context attributes for the i-th situation involved in the situation algebra expression. The number of orthotopes and the number of involved context attributes for the resulting situation is referred to as L_{res} and N_{res} respectively.

If the situation is converted from original CST representation into the orthotope-based representation, then the number of the orthotopes in the converted situation is equal to the number of combinations of intervals (see the expressions (24) and (25) and algorithm 5.1 for the proof). It allows expressing the formula (61) as the formula (62).

$$O(\sum_{i=1}^{K} N_i * (\prod_{j=1}^{N_i} r(i,j)) + N_{res} L_{res} + L_{res}) \quad (62)$$

In formula (62) the term $r(i,j)$ refers to the number of intervals for the i-th situation over j-th context attribute (within the situation). Formula (62) provides the final result of the complexity estimation. The formula (63) is used just for illustration purposes, in order to provide rough estimation of the total complexity order. Expression (63) is derived from formula (62) using the assumptions that the number of context attributes is the same for every involved situation (we refer to it as N), and the number of intervals over any context attribute is the same for any of the involved situations: $N_i = N_{avg}$, r(i,j) = r, i=1…K, j=1…N_i) We also assumed that the number of intervals is the same for every context attribute in the resulting situation, and refer to that number as r_{res}.

$$O(K * N_{avg} * {r_{avg}}^{N_{avg}} + N_{res} * {r_{res}}^{N_{res}} + {r_{res}}^{N_{res}}) \quad (63)$$

The algorithm is supposed to work offline, and it makes the complexity requirements milder. However, the formula (63) very roughly shows that the worst case rate of growth of the straightforward verification can be roughly estimated as $K*N_{avg}*exp(N_{avg})$ + $(N_{res}+1)*exp(N_{res})$ function of the involved context attributes and the number of involved situations. The complexity still remains tractable for the realistic situations (which are usually defined over small number of context attributes), but the fast verification algorithms are considered to be the high priority item of the future work.

In this section we theoretically calculated the complexity of the verification process and tested it practically. Formula (62) contains the final result of the verification complexity calculation, and it finalizes the complexity evaluation of the algorithm.

7 Discussion and Related Work

In this article we presented the approach for formal verification of situation models. In the following section we are going to discuss the several aspects of the proposed approach, compare the approach with related work and provide some discussion.

7.1 Formal Verification in Pervasive Computing

The techniques of formal verification have received some attention of the pervasive computing research community, and various aspects of pervasive computing systems were enhanced with applicable verification methods.

Cardelli and Gordon [CG00] proposed the ambient logic – a specially designed calculus that can describe multiple moving and interacting entities. That approach was improved and extended in subsequent works. For example, Coronato and Pietro [CP10] used ambient logic techniques to specify pervasive healthcare applications. Also Coronato and Pietro [CP11] proposed the extensions of ambient calculus and ambient logic. The main difference between the ambient logic-based approaches and our article is that the specification aspects under verification are different. Ambient logic based approaches view the pervasive system as the set of interacting agents, and thus aim to verify the spatial and temporal interactions between devices, processes and services. In contrast, our article proposes a solution to verify the integrity of the context model – the underlying formal representation of internal and external environment.

Ishikawa et. al. [IS09] proposed methods to verify and specify pervasive computing systems. Their verification approach is based on the event calculus formalism. Event calculus allows representing the behavior of the system and the assumptions about the behavior, and then formally verify, whether the behavior matches the assumptions. The article [IS09] addresses the interaction of pervasive computing system with an external environment. In contrast, our method allows the verification of system understanding of internal and external environment, and that understanding is the root cause of system behavior. Therefore, our approach allows deeper insight into the context awareness and situation awareness, its problems and risks.

To summarize, the related work did address the verification of internal communications [CG00, CP10, CP11] and user interactions [IS09] in pervasive computing systems. To the best of our knowledge our article is the first to address the verification of the integrity, consistency, non-contradiction and adequacy of the context model – the system understanding of internal and external environment.

7.2. Specification of Situation Relationships

The verification approach presented in this article uses the expected situation relationship as the part of the input information. Those relationships can be obtained using expert knowledge and common sense, but there are multiple solutions to produce the relationships automatically. Different approaches to situation awareness focus on the relationships between different situations rather than on inferring the situations from low-level context (e.g., sensor) data. Those approaches supply the properties for the formal verification mechanism. The approaches rely on some low-level inference of the atomic facts, events and situations, but on top of that they provide powerful tools to assert and facilitate the verification of various relationships between the situations.

The articles [DS07, ES07, WZ04] proposed to use the ontologies to reason about the situations. Ontologies provide the powerful tools to represent and reason about the relationships between different entities, facts, events and situations. Still many ontologies for representing the context work on the high-level context awareness level and consider the inference of low-level facts as being out of scope of the work. Therefore, ontology-based solutions require the low-level inference model, and that model should be consistent with the ontology itself. The consistency of the low-level model and the ontology can be verified using the methods presented in this article.

Ranganathan et. al. [RA04] used the predicate logic to define the events and situations. Moreover, the predicate logic was used to define the relationships between situations, such as mutual contradiction. However, those relationships were used not for the verification, but mainly for addressing context uncertainty.

The articles [AL08][GG06] proposed the solutions based on temporal logic. Logic-based expressions often can be the property under verification. However, working with the temporal logic is the part of future work. Neither the context spaces theory model, nor the situation algebra do not involve any timing or sequential dependencies. Sequential and timing dependencies can be analyzed using the concept of the trajectory in the application space.

7.3. Situation Modeling

Various ways of defining situations have received considerable amount of attention from pervasive computing research community. A number of methods of situation specification either originate in the context spaces theory, or have some degree of similarity with context spaces theory situation awareness approach.

Delir et. al. [DZ08] proposed improved situation specifications for the context spaces approach. Instead of step functions, the authors suggest to use standard fuzzy logic based contribution functions (like trapezoid). This approach leads to more natural and flexible situation specifications. However, the use of step functions as contribution functions also have some benefits. For example, the verification method proposed in this article heavily relies on the fact that contribution function is a step function, and therefore the confidence levels remain constant within certain orthotopes.

The papers [BI04][KK07][KK08][MB04][TI04] use naïve Bayesian approach to define the situations and activities. Naïve Bayesian approach assumes the independence of various context features, having the situation occurrence/non occurrence as a fact. It can be viewed as somewhat similar to the approach of context spaces theory, where the situation has independent contributions from various context attributes. In both cases the independence assumption ensures memory efficient and computationally tractable situation specifications, but in turn can lead to limited flexibility. The main difference in our approach to situation specification and the Bayesian approach is the difference in semantics. The Bayesian approach is purely probabilistic – it assumes that in underlying real world the situation is either occurring or not, and provides the method to evaluate the probability of occurrence. Context spaces theory, in contrast, uses fuzzy logic semantics, where the situation can occur with some confidence level, and it does not mean probability. For example, if the noise level is 50 dB, the confidence level of situation *Noisy* can be evaluated as 0.7, but stating that it is noisy with probability 70% is semantically wrong. Another serious difference is that the context spaces theory features specification-based approach, where the situations are defined manually, while the naïve Bayesian approach is better suited for learning the situation.

Anagnostopoulos et. al. [AN06] proposed the advanced situation awareness framework that leads the context reasoning from the low-level situation representation to properly acting on the situation awareness results. On the lowest levels the situation was inferred as the conjunction of multiple context features (see formula (64), quoted from [AN06]).

$$\bigwedge_{i=1}^{N} context(x_i, user) \rightarrow IsInvolvedIn(Situation, user), \quad N > 1 \tag{64}$$

Similar to the context spaces theory, the approach [AN06] to situation definition utilizes the contribution from multiple context features. However, the context spaces theory allows working with confidence levels and representing unequal importance of the different context features for the situation.

7.4. Geometrical Metaphors for Context Awareness

Our article uses geometrical metaphors in order to represent the context and reason about the situations. The idea of collecting the entire set of low-level data into the single vector is straightforward. That vector can be viewed as a point in a multidimensional space. However, many context reasoning approaches immediately advance further to higher level context awareness, and do not take the full advantage of spatial representation.

Anagnostopoulos et. al. [AM05] modeled the context as a multidimensional space, the current conditions of the context as the point in the space, and utilized this representation for context prediction. The authors predicted the context using the extrapolation of context trajectory in the multidimensional space. However, that paper did not provide any concept of the situation and did not generalize the context data in any manner, and the generalization in terms of situations provides the mechanism to extract the most important information from the context. The context prediction approach similar to [AM05] was proposed for the context spaces theory by Padovitz et. al. [PL08b].

The article by Mayrhofer [Ma04b] utilized the geometrical metaphors and viewed context as a vector in a multidimensional space of context features. The clusters in the space of context features were considered to be the possible situations of interest. That solution allowed learning the situations and then manually labeling them in a meaningful manner. Also that approach provided the solid background for context prediction. However, clustering methods usually result in situation definition being a blackbox and the situations being human-unreadable. Learning-based and specification-based approaches to the situation definition can be combined within one system.

8 Conclusion and Future Work

In this article we propose, develop and evaluate the mechanism for formal verification of context model and situation definitions. Using the situation definitions in terms of low-level context features and the expected situation relationships as an input, the mechanism can either formally prove the compliance between the properties and the definitions, or identify the entire set of possible context states, that can serve as a counterexample that proves situation inconsistencies and exposes a contradictory context model.

As a part of verification method, we propose the novel situation model, an orthotope-based situation space, which provides sufficient flexibility to reason about a broad class of real-life situations and their relationships, retain the human-readable representation of the situation, and have a set of properties that are very important for the verification process. We develop, implement and evaluate the algorithm to represent arbitrary situation algebra

expression as an orthotope-based situation space. We develop, implement and evaluate the algorithm to check an orthotope-based situation space for emptiness and detect the entire set of counterexamples.

The final contribution of this article is the context model verification method. The proposed mechanism can formally verify that the context model does not have internal contradictions, that it implies the necessary knowledge about the environment and that it does not contradict the real world facts. The verification method can detect the specification errors during the design of the context model, help an expert to properly fix the specification, and therefore improve the reliability of the system and reduce the risk of the pervasive system design error.

We consider the following directions of the future work:

1. **Fast verification algorithms**. The verification algorithm has roughly exponential dependency on the number of involved context attributes and, therefore, will highly benefit from the algorithms optimization.
2. **Verification of dynamic and temporal situation properties**. The properties that involve sequence or time (like "If the situation is PhoneCall, it means that there was a situation PhoneRings or Dialing before") are currently out of scope of the verification process. The ability to incorporate the temporal relationships into the verification significantly broadens the verification capabilities.
3. **Automated situation fix**. The current version of the verification approach verifies if there is a specification error and describes the context attributes that can lead to the context model inconsistency. However, the counterexamples only give the clue about how the specification error can be fixed. If the verification algorithm could provide suggestions for the situation specification fix on its own, it can facilitate the work of the expert.
4. **Combining verification and uncertainty**. The approach proposed in this article assumes that the context state is a particular point in a multidimensional space, and the specifications of the uncertainty are not provided. For different ways of uncertainty representation there should be either a proof that the verification approach is still applicable, or the amended verification algorithm.
5. **Context quality evaluation**. The expected relationships between situations can be viewed from different perspective. For example, the situation "SunnyDay & BlindsOpen" practically implies "LightLevelOK", but it is not implied by the model directly. These relationships can be asserted into the context model, and then be used to verify the consistency of the context state. For example, if the situations *SunnyDay(X) & BlindsOpen(X) & ¬LightLevelOk(X)* holds true, it means a problem with either of the sensors, that is involved in those situations. The algorithm that can identify the inconsistency and propose the solution to fix the context model can improve the run-time context awareness reliability and efficiency.
6. **Learning the situations**. Context spaces theory features the specification-based approach to the situation awareness. The ability to detect the possible presence of situation of interest, and suggest the specification of that situation, can significantly facilitate the work of the expert and improve the context model.

Chapter VII

Correctness Analysis and Verification of Fuzzy Situations in Situation Aware Pervasive Computing Systems

Based on:

1. Boytsov, A. and Zaslavsky, A. Correctness Analysis and Verification of Fuzzy Situations in Situation Aware Pervasive Computing Systems. Scientific report, 2013, 30p.

URL= http://pure.ltu.se/portal/files/42973133/BoytsovZaslavsky_FuzzyVerifReport.pdf, last accessed May, 08, 2013.

Foreword

This chapter addresses the research question 2 and continues the research direction described in chapter VI. Fuzzy logic complements situation awareness with versatile and robust situation concepts. This chapter proposes, proves and evaluates a novel verification algorithm for fuzzy situations. Chapter VII uses basic concepts of verification originating in chapter VI, but in order to apply verification principles to fuzzy situations a completely different algorithm is proposed, implemented and evaluated.

Correctness Analysis and Verification of Fuzzy Situations in Situation Aware Pervasive Computing Systems

Abstract. Context awareness is one of the central features of pervasive computing systems. From pervasive computing perspective a situation can be defined as external semantic interpretation of context. Situation awareness aims to infer situations out of context. Developing situation awareness is a challenging task, which can be significantly hampered by errors during design stage. In this article we propose a novel method for verification of fuzzy situation definitions. Fuzzy logic is a powerful mechanism for reasoning in pervasive computing systems and verification of situation models is a new method of formally ensuring correctness of context awareness and situation awareness. Verification is applied at the design time to check that definitions of situations are error-free. Verification approach allows developers to rigorously specify expected relationships between situations and then formally check that definitions of situations comply with expected relationships. If an error is found, then additional task is to find counterexamples - particular context attribute values, which can cause situation awareness inconsistency. Counterexamples provide additional insight into the cause of error and help repairing situation definitions. We also discuss a method to formalize requirements, as well as propose and formally prove the novel verification algorithm for fuzzy situation models. Last, but not least, we analyze theoretical and practical complexity of the proposed solution.

Keywords: context awareness; situation awareness; fuzzy logic; fuzzy situation inference; situation algebra; verification

1 Introduction

Context awareness is one of the foundational principles of pervasive computing. Context is the key characteristic of every pervasive computing system and, according to predictions [GP09], by 2015 context will be as influential in mobile consumer services, as search engines are influential to the web.

Situation in pervasive computing can be viewed as a higher level of generalization of context. For example, multiple wearable accelerometers on user's arms and legs produce enough information to detect situations like "user walking", "user running", "user standing" or "user sitting" (see [BI04]), in the room the data from noise sensors, movement sensors and appliance usage sensors can be generalized into situations "nobody in the room", "one person in the room" or "several people in the room" (see [DP11]), blood pressure sensory data in mobile healthcare can be generalized into situations "hypertension", "hypotension" and "normal pressure" (see [DZ08]). Situation awareness functionality extracts most general relevant information from context and provides it in a clear manner to the applications.

Practically used situation modeling methods include Naïve Bayesian approach [BI04][KK07], fuzzy logic [AN06][DZ08], belief function theory [DP11], context spaces [PL08a], neural networks [YW08], and many more (see [YD12] for a comprehensive survey). Situation models can be either learned from labeled [BI04][KK07] or unlabeled [HM06][Ma04a][Ma04b] data, or designed manually [DZ08][DP11] using the expert knowledge of the subject area. Situation reasoning result might be, for example, a Boolean

value (whether the situation occurs or not), probability that the situation is occurring, fuzzy level of confidence.

Design of a situation aware system is a complex and error-prone task. Developing situation definitions manually may be hampered by expert errors. Errors in training data, overfits or underfits, as well as choice of an unsuitable learning approach can hamper automated learning of situation models. Incorrect situation awareness results are transferred to applications, and in turn it leads to improper adaptation actions. For example, if the situations "one person in the room" and "two people in the room" are triggered simultaneously, it might be a result of sensor error, but it as well might be a result of an improper generalization, i.e. it can happen if the sensor readings are not translated into situations correctly. Or, for example, definition mistake might result in triggering together situations "user sitting" and "user walking" even if the sensor error has no impact. As another example, situation "driving" should imply situation "in the car", and if for some reliable sensor data it is possible to have the "driving" situation, but not "in the car" situation, this points to a definition mistake. Consider also one more example: a smart office, where there are two situations of interest associated with each workplace. One situation is "conditions acceptable for work", which means that the workplace environment (including light level and noise level) is sufficiently good to continue working at that workplace. The second situation is "light malfunctions", and it is triggered if the light is on, but the level of light is still insufficient to continue: the lamp may produce too dim light or be just off. According to their meaning, those situations should not co-occur – by definition "light malfunctions" implies insufficiency of light level, while "conditions acceptable" implies that all the workspace parameters are sufficient. Triggering both situations means contradictive situation awareness result, which might propagate further and lead to erroneous adaptation actions. This scenario will be a motivating example throughout the article. This example was inspired by the example from our previous article [BZ12b], but in this work it is redeveloped for fuzzy situation inference and analyzed by completely different approach.

In order to avoid definition errors, situation models often undergo extensive testing: developers thoroughly check that the situation awareness recognize proper situation for certain sensor inputs. Sensor inputs can be from real sensors or imitated sensors in simulated environment. However, the capabilities of testing are limited. The sensor values, which trigger certain problems, might as well not be considered during the testing. Verification is an acknowledged opportunity of reducing the amount of errors in protocols and programs. Usually verification ensures that the program has a certain property, like "if a request comes, it will be processed" (see [CG99] for more details on verification of software). In order to prove that a program has certain property, corresponding assertion under verification has to be expressed in a formal manner (e.g. as a temporal logic formula). Verification procedure then rigorously proves that for the given program the assertion is always true, or finds counterexamples – precise description of cases, where assertion fails. In pervasive computing some articles proposed methods to verify behavior rules [AC09][IS09] or multi-agent interaction aspects [CP10][CP11]. However, the plausibility of situation definition verification is often overlooked.

Some situation awareness-related articles, including our previous article [BZ12b], defined the problem of verification of situation models, i.e. detection of errors in the formulas of the situations. This article in comparison with [BZ12b] proposes novel methods to verify the situations, which are defined using fuzzy logic.

The paper is structured as follows. Section 2 introduces the background of the work. It provides an overview of spatial representation of context, introduces fuzzy situation

inference and recaps the concept of verification of situation models. Section 2 also contains multiple examples, which emerge into the motivating scenario – a running example, which we are going to use throughout the article in order to illustrate the proposed approach. Sections 3 proposes and proves step-by-step algorithms, which allow verification and, hence, detection of errors in the definitions of fuzzy situations. Section 4 describes evaluation results and provides complexity analysis of the proposed approach. Section 5 provides description of related work and discussion of the achieved results. Section 6 proposes future work directions and concludes the paper.

2 Background

This section provides the background to the challenges of verifying fuzzy situation models. It starts with discussing spatial representation of context, proceeds with fuzzy situation inference and introduces the concept of situation verification. The motivating example, which is briefly mentioned in the introduction, is built throughout the entire background section. The section is concluded with fully formalized and detailed motivating scenario that will be used as an illustration in the rest of the paper.

2.1 Spatial Representation of Context

Spatial representation of context emerges from a relatively straightforward idea of collecting all context attributes into a single vector of values. Context attributes include relevant sensor readings and the values derived from them. For example, in a mobile device with accelerometer and orientation sensor, a vector of context values can include readings of accelerometer, readings of orientation sensor, and absolute acceleration values, calculated from relative acceleration and orientation. All possible context vectors define multidimensional space of possible context.

Representing context as a vector ensures clarity and allows efficient situation awareness using subspaces in multidimensional space [BZ12b][DZ08][Ma04a][Ma04b][PL08a]. Also it enables context prediction based on extrapolation of context trajectory [AM05][PL08a]. However, despite seeming simplicity, spatial representation of context contains some challenges related to missing sensor readings, non-numeric values, sensor uncertainty and the fact that sensor readings arrive at different time and may get outdated.

The terminology, which we are going to define in this section, is based on terms established in the articles [BZ12b][DZ08][PL08a]. Comparing to the background work, in this article the terms are defined in a rigorous manner, in order to use them for formal proof and analysis of the verification algorithms.

An example of multidimensional context space in figure 1 is related to previously mentioned motivating scenario. The figure depicts a simple context space for a workspace in a smart office. Current context is represented as a point in the multidimensional space. In figure 1 at the workplace there is currently luminance of 500 Lx, noise level of 30 dB and the light switch is on. Those values can be raw sensor readings, or they can be the result of sensor data processing.

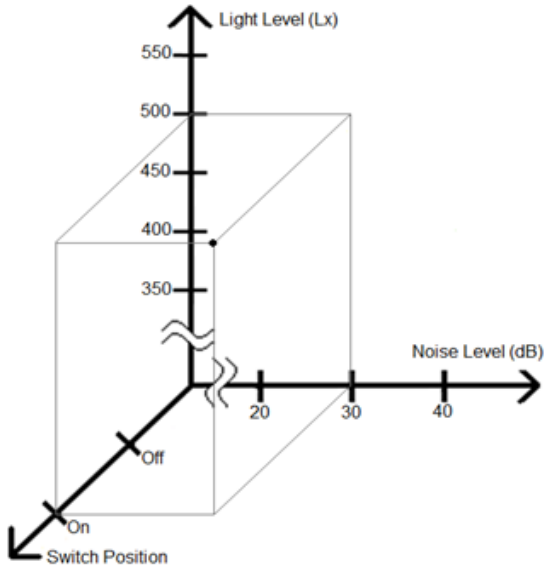


Fig.1. Example of Spatial Representation of Context

A multidimensional space, like in figure 1, is referred to as *application space* or c*ontext space* [PL08a]. Axes in the multidimensional space of figure 1 are context attributes. Formally *context attribute* is defined as domain of values of interest [PL08a]. Context attributes can as well be non-numeric: weather context attribute can have values like "sunny", "rainy" or "cloudy", tap valve context attribute can have values "open" and "closed", any appliance can be "On" or "Off". Some context attributes can have both numeric and non-numeric values. Those context attributes are referred to as mixed. For example, air conditioner can be configured to maintain certain temperature or it can be just off. Mixed context attributes allow graceful integration of missing context information: if some parameter is unknown, then the value for corresponding context attribute can be set to the special non-numeric value *Undefined* [BZ12b]. In order to simplify the example, in figure 1 *Undefined* values are omitted and context attributes *NoiseLevel* and *LightLevel* are considered to be numeric.

Context attribute value is a value of context attribute taken at a certain time [BZ12b]. Figure 1 shows that at present the value of *NoiseLevel* context attribute is 30 dB, the value of *LightLevel* context attribute is 500Lx and the value of *SwitchPosition* context attribute is *On*. In reality context attribute value takes uncertainty into account, but the questions of sensor uncertainty are out of scope of this article. Therefore, for the purpose of this article context attribute value is viewed just as a value on context attribute axis.

A point in the multidimensional context space is referred to as a *context state* [PL08a]. Therefore, a context state is a vector of context attribute values. The set of all possible context states will be referred to as $\mathbb{St}$. The fact that some entity X is a context state is formally expressed as $X \in \mathbb{St}$. Any context state *X* can be described like $\{c_1=x_1, c_2=x_{2,\dots}\}$, where c_i is a name of a particular context attribute. For example, the context state in figure 1 can be denoted as {*LightLevel*=500, *NoiseLevel*=30, *SwitchPosition*=*On*} or just {500,30,*On*}, if the order of context attributes in a vector is pre-defined.

2.2 Fuzzy Situations

Fuzzy logic is extensively used for context awareness and situation awareness purposes [AG10][AN06][CX05][DZ08][MS02]. In this article we use fuzzy situation format that was proposed in the paper in [DZ08]. Fuzzy situation is a versatile fuzzy logic-based situation awareness concept, and the algorithms developed for fuzzy situations can be applied to many other situation awareness techniques with minor modifications or no modifications at all.

Situation in pervasive computing is "external semantic interpretation of sensor data" [YD12]. From context reasoning perspective situation is a formula, which takes context state as an input and produces reasoning result as an output [BZ12b]. Reasoning result is often a numeric value, representing either probability of the situation occurrence or confidence in the fact that the situation is occurring. Also reasoning result can be Boolean, representing whether the situation occurs or not. Situations with Boolean reasoning results can be viewed as subspaces of the context space.

Fuzzy situation is a situation of a special format, which is presented in expression (1). The format of fuzzy situation was proposed in the paper [DZ08] in order to simplify the design of situations, prevent possible design mistakes and ensure readability.

$$Situation(X) = \sum_{i=1}^{N} w_i * \mu_i(x_i) \quad (1)$$

The term *Situation(X)* refers to the result of situation reasoning. This result is referred to as *certainty* or *confidence value*. Originally certainty was defined as numeric [DZ08]. In this article we augment it with a special value *UD*, which stands for undefined. It allows accounting for missing sensor readings and handling the cases when there is not enough information to reason about the situation.

Input parameter *X* in expression (1) is a context state. Vector X includes a set of values of relevant context attributes. Those values are referred to as x_i, where i = 1…N and N is the number of relevant context attributes.

In formula (1) coefficient w_i is the weight of i-th context attribute contribution to the final confidence. All the weights sum up to one. The function $\mu_i(x_i)$ is a membership function, which defines the contribution of i-th context attribute value. Every membership function depends on the value of only one context attribute. The term membership function comes directly from fuzzy logic - contribution is determined by the degree of belonging of a context attribute value to a specially designed fuzzy set [HM93]. The most popular shapes of membership functions are depicted in the figure 2 [DZ08][HM93]. In case if context attribute is non-numeric or mixed, there is a fixed value of the membership function associated with every possible non-numeric value.

For this article we enforce only the following requirements on membership functions:

1. Membership functions should be continuous in the numeric part of respective context attributes.
2. Derivatives of membership functions should be piecewise constant.

Given requirements encompass all functions depicted in figure 2, as well as all polygonal curve shaped membership functions. In a very general form we assume that a membership function over a numeric context attribute (or over numeric part of a mixed context attribute) is defined according to formula (2).

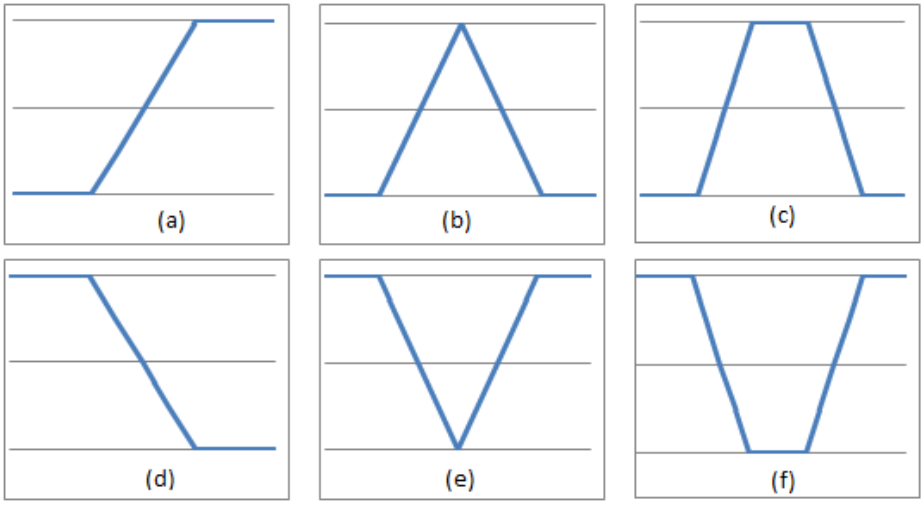


Fig. 2. Popular shapes of a membership function. (a),(d) Step membership function; (b),(e) Triangle membership function; (c),(f) Trapezoid membership function.

$$\mu(x) = \begin{cases} a_1 * x + b_1, x \in (-\infty, p(1)] \\ a_2 * x + b_2, x \in [p(1), p(2)] \\ \dots \\ a_L * x + b_L, x \in [p(L-1), p(L)] \\ a_{L+1} * x + b_{L+1}, x \in [p(L), +\infty) \end{cases} \quad (2)$$

In formula (2) the points p(1)…p(L) are referred to as breakpoints. The number of breakpoints L usually varies from 2 (in step function – figures 2a and 2d) to 4 (in trapezoid – figures 2c and 2f). For particular membership function $\mu_i(x)$of i-th context attribute the breakpoints will be referred to as p(i,1)…p(i,L_i), where L_i is the number of breakpoints in the membership function $\mu_i(x)$.

Additional requirement for the membership function is continuity. Note that the intervals in formula (2) overlap on the boundaries. Compliance with formula (3) ensures that there are no contradictions. It shows that different parts of the piecewise linear functions connect at the breakpoints.

$$\begin{cases} a_1 * p(1) + b_1 = a_2 * p(1) + b_2 \\ a_2 * p(2) + b_2 = a_3 * p(2) + b_3 \\ \dots \\ a_l * p(L) + b_L = a_{L+1} * p(L) + b_{L+1} \end{cases} \quad (3)$$

A special case of L=0 is acceptable. In practice it is often a constant membership function with zero value. It does not contradict formulas (2) and (3), and it acts as a stub membership function.

Non-numeric values can be incorporated into membership function by assigning membership function value for every non-numeric parameter. An example is depicted in formula (4).

$$\mu(x) = \left[UD, x \in \{\overset{\cdots}{Undefined}\} \right. \quad (4)$$

For examples of fuzzy situations, refer to the context space depicted in figure 1. In that context space we can define two situations. A first situation under consideration is the situation, reflecting whether the workplace conditions are sufficient for normal work. For the purpose of illustration we consider only light level and noise level at the workplace. Membership functions of light and noise levels are presented in figure 3 and formalized into formula (5). In order to distinguish between membership functions of different situations over the same context attribute, the superscript over μ contains abbreviation of the situation name (CA stands for *ConditionsAcceptable*).

$$\mu^{CA}_{\text{Light}}(\text{LightLevel}) = \begin{cases} 0, \text{LightLevel} \leq 350 \\ \frac{LightLevel-350}{150}, \text{LightLevel} \in [350,500] \\ 1, \text{LightLevel} \geq 500 \end{cases} \quad (5)$$

$$\mu^{CA}_{\text{Noise}}(\text{NoiseLevel}) = \begin{cases} 1, \text{NoiseLevel} \leq 40 \\ (60 - NoiseLevel)/20, \text{NoiseLevel} \in [40,60] \\ 0, \text{NoiseLevel} \geq 60 \end{cases}$$

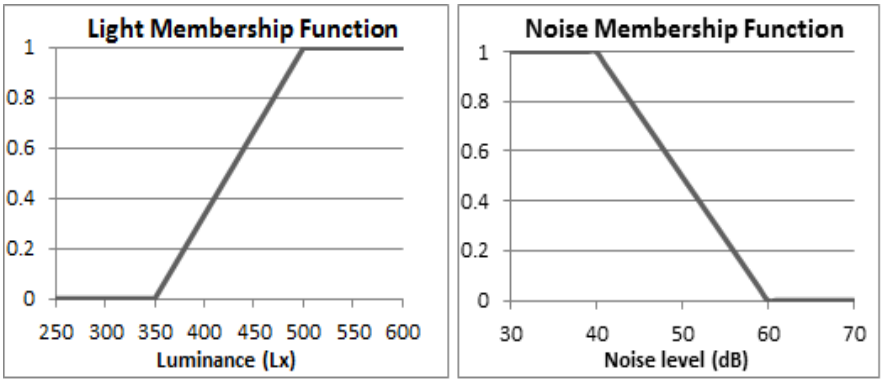


Fig. 3. Membership functions of *ConditionsAcceptable* situation

Light level and noise level are equally important characteristics at the workplace and, therefore, they are assigned the same weight. We define situation *ConditionsAcceptable* according to expression (6).

$$\text{ConditionsAcceptable(X)} = 0.5 * \mu^{CA}_{\text{Light}}(\text{LightLevel}) + 0.5 * \mu^{CA}_{\text{Noise}}(\text{NoiseLevel}) \quad (6)$$

Another situation under consideration is whether the light is malfunctioning. The problem is detected if lamps are on, but still provide insufficient light. For example, the lamps can be too dim due to internal malfunction, or they can become unpowered due to wire problems. The contributing parameters are position of light switch and light level. Contributions are depicted in figure 4 and formalized into expression (7). Superscript on top of membership function distinguishes the situation (LM stands for *LightMalfunctions*).

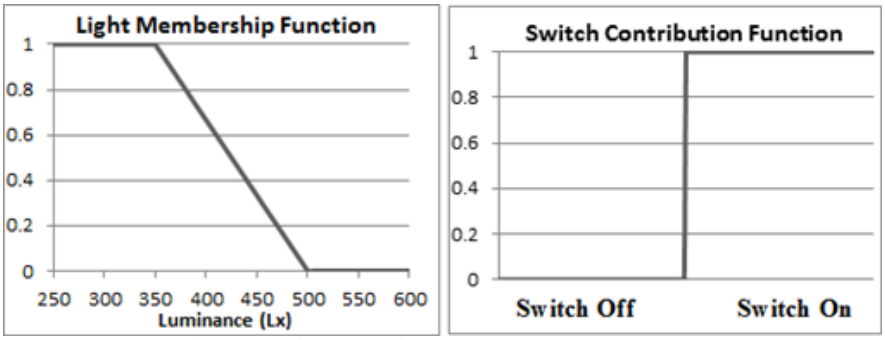


Fig. 4. Membership functions of *LightMalfunctions* situation

$$\mu^{LM}_{\text{Light}}(\text{LightLevel}) = \begin{bmatrix} 0, \text{LightLevel} \leq 350 \\ (500 - LightLevel)/150, \text{LightLevel} \in [350,500] \\ 1, \text{LightLevel} \geq 500 \end{bmatrix} \quad (7)$$

$$\mu^{LM}_{\text{Switch}}(\text{SwitchPosition}) = \begin{bmatrix} 1, \text{SwitchPosition} \in \{On\} \\ 0, \text{SwitchPosition} \in \{Off\} \end{bmatrix}$$

Final formula of the situation is depicted in expression (8).

$$\text{LightMalfunctions(X)} = 0.5 * \mu^{LM}_{\text{Light}}(\text{LightLevel}) + 0.5 * \mu^{LM}_{\text{Switch}}(\text{SwitchPosition}) \quad (8)$$

The situations, defined in expressions (5)-(8), will be used throughout this article as running examples. Several more definitions are necessary to proceed to verification of situation models.

Fuzzy situation inference uses *certainty threshold* to decide whether the situation occurs or not. If the certainty reaches the threshold, situation is counted as occurring. If the certainty is below the threshold or the certainty is undefined, then occurrence is not claimed.

Situation algebra was developed in order to reason about relationships between situations [PL08a]. Situation algebra procedures resemble Zadeh operators [Za65]. Details are provided in expression (9).

$$\begin{aligned} AND: (A \,\&\, B)(X) &= min(A(X), B(X)) \\ OR: (A \mid B)(X) &= max(A(X), B(X)) \\ NOT: (\neg A)(X) &= 1 - A(X) \end{aligned} \quad (9)$$

If the certainty value of *A(X)* or *B(X)* is undefined, then the result of situation algebra operation is undefined as well. The notations like (A | B)(X) and A(X) | B(X) are equivalent and just represent different styles.

Next section uses the definitions from this section and on their basis defines the essentials of verification of situation definitions.

2.3 Verification of Context Models and Motivating Scenario

Boytsov and Zaslavsky [BZ12b] analyzed and formalized possible relationships between situations and utilized them for context model verification. One of the main outcomes of the article [BZ12b] is that non-temporal situation relationships can be expressed as assertions of emptiness for some situation algebra expressions. The relationships between situations, as well as the corresponding assertions are presented in formula (10).

$$\begin{aligned}
&\text{Generalization: } \nexists X \epsilon \mathbb{S}\mathbb{t}, \\
&\quad (LessGeneralSituation \& \neg MoreGeneralSituation)(X) \geq th \\
&\text{Composition: } \nexists X \epsilon \mathbb{S}\mathbb{t}, \\
&\quad (Situation \ \& \neg Component1 \ \& \neg Component2 \ldots \& \neg ComponentN)(X) \geq th \\
&\text{Dependence: } \nexists X \epsilon \mathbb{S}\mathbb{t}, \ (Situation \ \& \neg DependsOn)(X) \geq th \\
&\text{Contradiction: } \nexists X \epsilon \mathbb{S}\mathbb{t}, (Sit1 \ \& \ Sit2)(X) \geq th
\end{aligned} \tag{10}$$

The term *th* in formula (10) denotes certainty threshold. Expressions (10) are assertions of *emptiness with respect to the threshold*. *Empty* situation algebra expression [BZ12b] is an expression that cannot reach certainty threshold for any input context state. Therefore, empty expression will never be recognized as occurring.

In formula (10) *generalization* means that one situation is less general than the other, e.g. a situation *InTheLivingRoom* is less general then a situation *AtHome*. Corresponding assertion states that less general situation should not occur without more general situation, or, rigorously, for no context state should the certainty of having less general situation and not more general situation reach the certainty threshold. *Composition* means that the situation is built from several components. The assertion states that for no context state should the situation be recognized without any of its components. For example, situation *InTheHouse* is composed of situations like *InTheLivingRoom*, *InTheHall*, *InTheBathroom*. *Dependence* means that one situation is a prerequisite to another (e.g. situation *UserWatchingTV* depends on situation *TVisOn)*. Contradiction means that situations should not occur simultaneously, and the motivating scenario is a good example of it. More sophisticated assertions can be verified by using situation algebra expressions as arguments for the formulas (10).

A context state, for which the assertion under verification is false, is referred to as a *counterexample*. If a developer needs to know, whether the assumption is satisfied or not, it is enough to find at least one counterexample or prove that there are none. Finding as many counterexamples as possible might provide better insight on how to fix the definition error, but in turn searching for more counterexamples might require more time and computational resources.

We demonstrate the approach by applying it to illustrative example. The motivating scenario is built on top of the example context and situations, which were defined in earlier sections. The example was inspired by [BZ12b], but we use different and more advanced concept of a situation, and it results in more realistic situation representation. Moreover, different concept of a situation results in entirely different verification algorithm, which is proposed and proved in this article.

Consider the context space, depicted in figure 1. It has three context attributes: numeric attributes *NoiseLevel* and *LightLevel* and non-numeric context attribute *SwitchPosition*. The situations *LightMalfunctions* and *ConditionsAcceptable*, are defined in expressions (6) and (8) respectively. As we mentioned in the introduction, the situations *ConditionsAcceptable* and *LightMalfunctions* should not co-occur: *ConditionsAcceptable* implies illuminance

sufficiency, while *LightMalfunctions* implies illuminance insufficiency. In terms of situation relations (10), *ConditionsAcceptable* and *LightMalfunctions* are in a contradiction. The value 0.7 is frequently chosen as confidence threshold in practice, and we are going to choose it for the motivating example. As a result, assertion (11) should be maintained.

$$\nexists X \in \mathbb{St}, \quad (LightMalfunctions \ \& \ ConditionsAcceptable)(X) \geq 0.7 \tag{11}$$

Assertion (11) states that for no context state the confidence level of the situation algebra expression *ConditionsAcceptable&LightMalfunctions* exceeds the threshold, i.e. for no context state the two mentioned situations are triggered together.

This section provided rigorous formalization of the necessary terms and defined a motivating scenario. The next section proposes verification algorithm for fuzzy situations and uses verification of assertion (11) as a running example.

3 Verification of Fuzzy Situations

In this section we propose a method of emptiness check for a situation algebra expression that involves fuzzy situations. Section 3.1 describes additional assumptions and notation agreements. Section 3.2 proposes a general method to utilize DNF representation of the verified assertion. Section 3.3 describes the method to handle non-numeric and mixed context attributes. Sections 3.4 and 3.5 propose the method to find subspaces of context space, where all the involved situations are linear functions. Section 3.6 proposes a method to find the maximum confidence values within those subspaces. Section 3.7 summarizes the verification approach. Every step of verification is illustrated in details using the motivating scenario from section 2.3.

3.1 Additional Assumptions

In order to simplify further proofs and analysis, we make several additional assumptions. Those assumptions do not result in any loss of generality, and can be viewed more as notation agreements.

Assumption 1. All the mentioned situations are defined over the same set of context attributes.

This assumption simplifies the proofs without reducing generality of the methods: if needed, additional context attributes with zero weight and constant zero membership functions can be introduced into any situation. In the motivating scenario we can rewrite the situations in the following manner (expressions (12) and (13)).

$$\begin{aligned}&\text{ConditionsAcceptable(X)} = 0.5 * \mu^{CA}_{\text{Light}}(\text{LightLevel}) + \\ &\quad +0.5 * \mu^{CA}_{Noise}(\text{NoiseLevel}) + 0 * \mu^{CA}_{\text{Switch}}(SwitchPosition) \\ &\mu^{CA}_{\text{Switch}}(SwitchPosition) = 0\end{aligned} \tag{12}$$

$$\begin{aligned}&\text{LightMalfunctions(X)} = 0.5 * \mu^{LM}_{\text{Light}}(\text{LightLevel}) + \\ &\quad +0 * \mu^{LM}_{\text{Noise}}(\text{NoiseLevel}) + 0.5 * \mu^{LM}_{\text{Switch}}(SwitchPosition) \\ &\mu^{LM}_{\text{Noise}}(\text{NoiseLevel}) = 0\end{aligned} \tag{13}$$

Expressions (12) and (13) are equivalent to the definitions (6) and (8) respectively. In expressions (12) and (13) both situations, involved in the motivating scenario, are defined over the same set of context attributes: *LightLevel*, *NoiseLevel* and *SwitchPosition*. If a

context attribute influences at least one involved situation, it is added to all the other situations. Note that even if newly added context attribute is *Undefined* (e.g. due to unavailable sensor), the membership function is still zero.

Assumption 2. Context space has only context attributes, which belong to at least one situation. Only those context attributes influence certainty levels, and, therefore, only those context attribute values determine whether any given context state is a counterexample or not.

In the motivating example the only relevant context attributes are *LightLevel*, *NoiseLevel* and *SwitchPosition*. Other context attributes do not influence assertion (11) and, therefore, are omitted from context space by this assumption.

Assumption 3. Context attributes within context state are ordered and numbered. It will ensure that when referring to i-th element of input context state, the same context attribute is implied for different situations. According to assumption 1, for all the situations the list of context attributes is the same. Therefore, choosing any arbitrary order will satisfy the assumption.

In the motivating scenario we define the following order: {*LightLevel*, *NoiseLevel*, *SwitchPosition}*. For example, the point from figure 1 is {500,30,On}. Other context attribute values are omitted according to assumption 2.

Further sections propose and prove the verification algorithm, and the derivations imply the assumptions 1-3.

3.2 Utilizing DNF representation

Disjunctive normal form (DNF) [RN06] is a way to represent logical expression as a disjunction of conjunction clauses. A generic example of DNF situation algebra expression is presented in formula (14).

$$(Sit1(X)\&Sit2(X)) \mid (Sit3(X)\&(\neg Sit4^*(X))) \mid ((\neg Sit5(X))\&Sit1(X)) \quad (14)$$

The whole expression (14) is a disjunction, where every disjunct is conjunction of single situations or their negations. The disjuncts in expression (14) are *Sit1(X)&Sit2(X)*, *Sit3(X)&(¬Sit4*(X))* and *(¬Sit5(X))&Sit1(X)*.

In formalized situation relations (10) expressions are naturally in DNF format. In the motivating example (11) expression is in DNF format as well. If needed, the following properties can assist DNF conversion. Those properties straightforwardly follow from formulas (9).

1. AND and OR are commutative:
   (A & B)(X) = (B & A)(X), and (A | B)(X) = (B | A)(X).
2. Distributive property holds for AND over OR:
   (A & (B | C))(X) = (A&B | A&C)(X).
3. DeMorgan laws do apply:
   a. ¬ (A | B)(X) = (¬A & ¬B)(X).
   b. ¬ (A & B)(X) = (¬A | ¬B)(X).

Lemma 3.2 shows how DNF representation can be utilized for verification purposes. The main idea is to find maximum achievable certainty for each disjunct, and obtain the context state where the maximum certainty is achieved. Lemma 3.2 shows that if counterexamples do exist, some of them will be at those context states.

Lemma 3.2. Any arbitrary DNF situation algebra expression is non-empty w.r.t. to some threshold, if and only if the maximum certainty value of at least one DNF disjunct is greater or equal to the threshold.

Proof. Consider an arbitrary DNF situation algebra expression *Expr(X)*, presented in formula (15).

$$Expr(X) = Disj_1(X) \mid Disj_2(X) \mid \ldots \mid Disj_N(X) \tag{15}$$

Every disjunct $Disj_i(X)$ is a conjunction of single situations or their negations, and for this lemma no further details are required. The chosen threshold is denoted as *th*.

Part 1. Sufficiency proof. The proof of sufficiency assumes that for at least one disjunct the maximum value reaches the threshold *th*, and derives that the whole expression is non-empty w.r.t. to that threshold. Let for some disjunct $Disj_k(X)$ the maximum achievable certainty value be *d*, which is not less than the threshold *th*. Let's also denote as X_k the context state, at which the maximum certainty value *d* is achieved by $Disj_k(X)$. It can be summarized in formula (16).

$$Disj_k(X_K) = d,\ d \geq th \tag{16}$$

Now let's find the certainty of the expression *Expr(X)* at context state X_K. The derivation is presented in (17). OR situation algebra operation is expanded according to definition (9).

$$\begin{aligned} &Expr(X_k) = Disj_1(X_k) \mid Disj_2(X_k) \mid \ldots \mid Disj_N(X_k) = \\ &= \max(Disj_1(X_k), Disj_2(X_k), \ldots, Disj_N(X_k)) = \\ &= \max(Disj_1(X_k), Disj_2(X_k), \ldots, Disj_{K-1}(X_k), d, Disj_{K+1}(X_k), \ldots, Disj_N(X_k)) \geq d \geq th. \end{aligned} \tag{17}$$

Summarizing the derivation (17), at the context state X_K the certainty of expression *Expr(X)* reaches the threshold *th*. Therefore the expression is not empty w.r.t. to the threshold *th*. It completes the sufficiency proof.

Part 2. Necessity proof. Necessity proof assumes that expression (15) is non-empty w.r.t. to the given threshold *th*, i.e. there exist some context state *X'* such that $Expr(X') \geq th$. The task is to prove that for at least one disjunct the maximum confidence value is greater or equal than the threshold *th*. Initial conditions for necessity proof can be summarized in expression (18).

$$\exists X' \in \mathbb{St},\ Expr(X') \geq th \tag{18}$$

For a proof by contradiction assume that the opposite is true – the maximum values of all disjunct are less than the threshold *th*. If this assumption results in a contradiction, it will prove that for some of the disjuncts the threshold will be reached. The assumption for proof by contradiction is summarized in expressions (19).

$$\max(Disj_i(X)) < th,\ i = 1 \ldots N \tag{19}$$

Consider the derivation (20). It follows from expressions (18) and (19) and situation algebra definitions (9).

$$\begin{aligned} th \leq\ & Expr(X') = Disj_1(X') \mid Disj_2(X') \mid \ldots \mid Disj_N(X') = \\ & = \max(Disj_1(X'), Disj_2(X'), \ldots, Disj_N(X')) \leq \\ & \leq \max(\max_x(Disj_1(X)), \max_x(Disj_2(X)), \ldots, \max_x(Disj_N(X))) < \\ & < \max(th, th, \ldots, th) = th \end{aligned} \tag{20}$$

Derivation (20) leads to summary $th < th$, which is a contradiction. It proves that assumption (19) is wrong and completes proof by contradiction. Therefore, for at least one disjunct the maximum confidence value should reach the threshold. **Q.E.D.**■

To summarize lemma 3.2, in order to check DNF situation algebra expression for emptiness it is sufficient to find the maximum certainty values of every disjunct separately and compare them against the threshold. If the threshold is exceeded for at least one disjunct, then the expression is not empty, and verification has detected an error. The context state, where maximum is achieved for that disjunct, is a counterexample. If all the maxima are below the threshold, then the situation definitions comply with the assertion under verification.

Subsequent sections propose and prove a detailed method to find the maximum value of any disjunct. Next section discusses the influence of non-numeric context values on the maximization task.

3.3 Handling Non-numeric Context Attribute Values

In section 3.2 we proved that for verification of DNF assertion it is sufficient to find maximum achievable certainty of every disjunct. The presence of non-numeric values in non-numeric or mixed context attributes poses a challenge to searching for maximum. A plausible solution is to reduce the task to multiple maximization tasks with numeric input.

For example, in the motivating scenario we can view two cases with purely numeric remaining context attributes: the case when *SwitchPosition=On* and the case when *SwitchPosition=Off*. The final task is to find two maximums for two numeric functions. Those functions are presented in expressions (21) (for *SwitchPosition=On*) and (22) (for *SwitchPosition=Off*). In expressions (21) and (22) the contributions of *SwitchPosition* are replaced by their exact values due to the fact that the value of *SwitchPosition* context attribute is fixed.

$$\min(0.5 * \mu^{CA}_{\text{Light}}(\text{LightLevel}) + 0.5 * \mu^{CA}_{\text{Noise}}(\text{NoiseLevel}), 0.5 * \mu^{LM}_{\text{Light}}(\text{LightLevel}) + 0.5) \quad (21)$$

$$\min(0.5 * \mu^{CA}_{\text{Light}}(\text{LightLevel}) + 0.5 * \mu^{CA}_{\text{Noise}}(\text{NoiseLevel}), 0.5 * \mu^{LM}_{\text{Light}}(\text{LightLevel})) \quad (22)$$

Mixed context attributes can be processed in similar manner. For example, consider a mixed context attribute *AirConditionerSetting*, which can have the values *Off* if the conditioner is off, *Undefined* when the settings are unknown (e.g. due to connection problems), or have a numeric value – the temperature set for the air conditioner. Three special cases need to be investigated in that case.

1. *AirConditionerSetting=Off*
2. *AirConditionerSetting=Undefined*
3. *AirConditionerSetting* $\in \mathbb{R}$

The general case of the described approach is proposed in operation 3.3. The goal of operation 3.3 is to reduce verification involving non-numeric context attributes to multiple verifications involving only numeric context attributes.

Operation 3.3. Consider a situation algebra expression *Expr(X)*. It is asserted that the certainty of *Expr(X)* does not reach the threshold, and in order to test it we need to find the maximum achievable certainty value. Among other context attributes, expression *Expr(X)* is defined over context attribute c_L, which can take non-numeric values $a_1 \ldots a_K$ (and, possibly, numeric values as well).

Proposition. In order to find maximum certainty of *Expr(X)*, it is sufficient to find maximum certainty for the following cases.

Subtask 1. Find maximum certainty of *Expr(X)* with the constraint $c_L = a_1$.

Subtask 2. Find maximum certainty of *Expr(X)* with the constraint $c_L = a_2$.

…

Subtask K. Find maximum certainty of *Expr(X)* with the constraint $c_L = a_K$.
Subtask K+1. Find maximum certainty with a condition that c_L is numeric (if c_L is a mixed context attribute).

Output. The largest of obtained maximums and the corresponding context state, where it is achieved.

The resulting expressions under maximization are denoted as *Expr(X* | $c_L = a_i$*)* or *Expr(X* | $c_L \in \mathbb{R}$*)*.

In the motivating scenario two resulting constrained optimization tasks are represented by the formulas (21) and (22). They respectively correspond to *(ConditionsAcceptable&LightMalfunctions)(X*|*SwitchPosition*=On*)* and *(ConditionsAcceptable&LightMalfunctions)(X*| *SwitchPosition*=Off*)*.

Proof. Let the maximum certainty value of *Expr(X)* be achieved at a context state *X'*. If in context state *X'* the value of context attribute c_L is equal to a_i, then this context state will be found while solving i-th subtask. If in context state *X'* the value of context attribute c_L is numeric, then it will be found while solving (K+1)-th subtasks. Therefore, if all the maxima for *Expr(X* | $c_L = a_i$*)* and *Expr(X* | $c_L \in \mathbb{R}$*)* are found, the global maximum can be obtained in a straightforward manner – it is the highest maximum value among the obtained maxima for tasks 1…K+1. Moreover, if any maximum for *Expr(X* | $c_L = a_i$*)* or *Expr(X* | $c_L \in \mathbb{R}$*)* exceeds the threshold, then it is already a counterexample.

Q.E.D.■

The main outcome of operation 3.3 is reducing the number of non-numeric parameters. On the first run of the operation, one context attribute c_L will be eliminated from the subtasks 1...K (just like *SwitchPosition* was eliminated from the input of expression *(ConditionsAcceptable & LightMalfunctions)(X)*). For the subtask K+1 the context attribute c_L will be reduced to numeric, and still the number of non-numeric inputs will be reduced by one. If there are many non-numeric or mixed context attributes involved, operation 3.3 should be applied recursively until all of those context attributes are processed. Next iteration is applied to all the subtasks, which emerged from previous iteration. After multiple iterations of operation 3.3 the resulting subtasks will contain only numeric parameters as inputs.

It should be specifically noted that after applying operation 3.3 the situations might no longer comply with definition (1). For example, in expression (21) the situation *LightMalfunctions* effectively becomes $0.5 * \mu^{LM}_{\text{Light}}(\text{LightLevel}) + 0.5$. Weights no longer sum up to 1, and a bias term is introduced. In order to proceed with the proof, we need a more general formula of a situation, which stays unaltered after applying operation 3.3. After the substitutions like $c_L = a_i$, which are part of operation 3.3, the involved situations comply with definition (23) instead of definition (1). Membership functions in the definition (23) are compliant with formulas (2) and (3).

$$Situation(X) = \sum_{i=1}^{N} w_i * \mu_i(x_i) + w_0 \qquad (23)$$

Fuzzy situations, which comply with the definition (1), do comply with the definition (23) as well. The term w_0 for them is equal to zero. Negated situations also comply with the definition (23), if the situation is defined according to expression (1) and negation is defined according to formula (9). For that case w_0 is equal to 1 and all the coefficients w_i change the sign comparing to original, non-negated situation. In definition (23) the coefficients w_i no longer have to be positive and no longer have to sum up to 1.

After one iteration of operation 3.3 the involved situations undergo substitution (24). Subscripts denote the subtask, and x_L denotes the value of expanded context attribute.

$$\begin{aligned} Situation_1(X) &= \sum_{i=1}^{L-1} w_i * \mu_i(x_i) + w_L * \mu_L(a_1) + \sum_{i=L+1}^{N} w_i * \mu_i(x_i) + w_0 \\ Situation_K(X) &= \sum_{i=1}^{L-1} w_i * \mu_i(x_i) + w_L * \mu_L(a_K) + \sum_{i=L+1}^{N} w_i * \mu_i(x_i) + w_0 \\ Situation_{K+1}(X) &= \sum_{i=1}^{N} w_i * \mu_i(x_i) + w_0, \text{ where } x_L \text{ is restricted to} \\ & \qquad \text{numeric values (if possible).} \end{aligned} \tag{24}$$

The (K+1)-th situation of formula (24) is directly compliant with definition (23). Let's prove that other situations of (24) remain compliant as well. Consider i-th expression of (24), where i is any integer value between 1 and K inclusively. The term $w_L * \mu_L(a_i)$ is a constant. Let's define the term w'_0 as $w_0 + w_L * \mu_L(a_i)$. Also let's redefine the terms in a following manner:

- $w'_j = w_j$ for j=1..i-1 and $w'_j = w_{j+1}$ for j= i...N.
- $\mu'_j = \mu_j$ for j=1..i-1 and $\mu'_j = \mu_{j+1}$ for j= i...N.

With redefined notation, i-th row of formula (24) can be rewritten as expression (25).

$$Situation_j(X) = \sum_{i=1}^{N-1} w'_i * \mu'(x_i) + w'_0 \tag{25}$$

Expression (25) is straightforwardly compliant with the definition (23). It proves that situations from formula (24) are all compliant with definition (23). To summarize, we have proven that after applying operation 3.3 the formulas of individual situations remain compliant with the expression (23).

In the motivating scenario for the situation *LightMalfunctions* the term w_0 is equal to 0.5 for the case SwitchPosition=*On* and is equal to 0 for the case SwitchPosition=*Off*. For the situation *ConditionsAcceptable* the term w_0 is equal to 0 in both cases.

The case where w_0 is undefined *(UD)* is a special case. It can appear in some subtasks when considering missing sensor values (see formula (4)). In that case the certainty of a situation is undefined. The whole conjunction of DNF expression, containing this situation is undefined, and so is the entire DNF expression. Whether to count undefined value as a counterexample or not is a matter of developer's choice, but in any case no further calculations are necessary for that subtask. Therefore, the case $w_0 = UD$ is trivial and from now and on we consider only the cases when w_0 is not *UD* (i.e. numeric), unless explicitly mentioned otherwise.

To summarize, the propositions, proven in this section, have the following implications for further proofs:

1. The search for maximum certainty can be performed separately for all possible combinations of non-numeric values. For mixed context attributes both possible numeric and non-numeric values should be taken into account. Operation 3.3 can generate a set of subtasks, which should be solved separately.

2. Within every subtask the input arguments are numeric only. Also within every subtask the situations, involved in the expression under verification, are compliant with definition (23). Compliance with definition (1) is not guaranteed.

The next section describes the search for maximum for every mentioned subtask, with respect to the implications summarized above.

3.4 Subspaces of Linearity – Single Situation

The extended definition of fuzzy situation (expression (23)) shows that the situation is a weighted sum of membership functions. In turn, expressions (2) and (3) show that membership functions are continuous piecewise linear functions, which depend only on a single context attribute. The input context state is numeric – otherwise the operation 3.3 should be applied first.

A membership function of any arbitrary context attribute contains a set of breakpoints, and between those breakpoints (as well as before the first breakpoint and after the last one) a membership function linearly depends on a single context attribute value. The whole situation formula becomes a linear function if the context state is bounded within Cartesian product of intervals between the breakpoints. Consider an illustration from the motivating example.

The situation *ConditionsAcceptable* can be correctly described both by definitions (26) and (27) for both cases *SwitchPosition=On* and to *SwitchPosition=Off*. Expressions (26) and (27) emerge from substituting expression (5) into expression (6).

$$ConditionsAcceptable(X) = \\ = \begin{cases} 0.5 * \mu^{CA}_{\text{Noise}}(\text{NoiseLevel}), \text{LightLevel} \leq 350 \\ \frac{1}{300} LightLevel + 0.5 * \mu^{CA}_{\text{Noise}}(\text{NoiseLevel}) - \frac{7}{6}, \text{LightLevel} \in [350; 500] \\ 0.5 * \mu^{CA}_{\text{Noise}}(\text{NoiseLevel}) + 0.5, \text{LightLevel} \geq 500 \end{cases} \quad (26)$$

$$ConditionsAcceptable(X) = \\ = \begin{cases} 0.5 * \mu^{CA}_{\text{Light}}(\text{LightLevel}) + 0.5, \text{NoiseLevel} \leq 40 \\ -\frac{1}{40} NoiseLevel + 0.5 * \mu^{CA}_{\text{Light}}(\text{LightLevel}) + \frac{3}{2}, \text{NoiseLevel} \in [40,60] \\ 0.5 * \mu^{CA}_{\text{Light}}(\text{LightLevel}), \text{NoiseLevel} \geq 60 \end{cases} \quad (27)$$

Expressions (26) and (27) can be merged in order to find the situation formula within various Cartesian product of intervals on *LightLevel* and *NoiseLevel* axis. The result of the merging, which straightforwardly follows from formulas (26) and (27), is presented in table 1. The formulas from table 1 apply to both cases *SwitchPosition=On* and *SwitchPosition=Off*.

The intervals in expressions (26) and (27) do overlap and, hence, the subspaces in table 1 do overlap as well. However, compliance with conditions (3) ensures continuity and protects from contradictions on the boundaries. As table 1 shows, for any context state, which belongs to several Cartesian products of intervals, it does not matter which line of table 1 to use for calculations – the resulting confidence value is the same.

For *LightMalfunctions* situation the formulas are presented in expressions (28) and (29). They correspond to the cases *SwitchPosition=On* and *SwitchPosition=Off* respectively. Note that those two cases correspond to different subtasks, generated by operation 3.3 (subtasks (21) and (22) respectively).

$$\text{LightMalfunctions(X)} = \\ = \begin{cases} 0.5, \text{LightLevel} \leq 350, \text{SwitchPosition} = \text{On} \\ -\frac{1}{300} LightLevel + \frac{13}{6}, \text{LightLevel} \in [350,500], \text{SwitchPosition} = \text{On} \\ 1, \text{LightLevel} \geq 500, \text{SwitchPosition} = \text{On} \end{cases} \quad (28)$$

Table 1. *ConditionsAccetable* – expanded formula.

Subspace	ConditionsAcceptable
$\text{LightLevel} \leq 350, \text{NoiseLevel} \leq 40$	0.5
$\text{LightLevel} \leq 350, \text{NoiseLevel} \in [40,60]$	$-\frac{1}{40}NoiseLevel + \frac{3}{2}$
$\text{LightLevel} \leq 350, \text{NoiseLevel} \geq 60$	0
$\text{LightLevel} \in [350; 500], \text{NoiseLevel} \leq 40$	$\frac{1}{300}LightLevel - \frac{2}{3}$
$\text{LightLevel} \in [350; 500], \text{NoiseLevel} \in [40,60]$	$\frac{1}{300}LightLevel - \frac{1}{40}NoiseLevel + \frac{1}{3}$
$\text{LightLevel} \in [350; 500], \text{NoiseLevel} \geq 60$	$\frac{1}{300}LightLevel - \frac{7}{6}$
$\text{LightLevel} \geq 500, \text{NoiseLevel} \leq 40$	1
$\text{LightLevel} \geq 500, \text{NoiseLevel} \in [40,60]$	$-\frac{1}{40}NoiseLevel + 2$
$\text{LightLevel} \geq 500, \text{NoiseLevel} \geq 60$	0.5

$$\text{LightMalfunctions}(X) = \\ = \begin{cases} 0, \text{LightLevel} \leq 350, \text{SwitchPosition} = \text{Off} \\ -\frac{1}{300}LightLevel + \frac{5}{3}, \text{LightLevel} \in [350{,}500], \text{SwitchPosition} = \text{Off} \\ 0.5, \text{LightLevel} \geq 500, \text{SwitchPosition} = \text{Off} \end{cases} \quad (29)$$

Expressions (28) and (29) are summarized in table 2.

As follows from table 1, there exists a set of subspaces within the context space, where the situation *ConditionsAcceptable* is linear. Every context state belongs to some subspace of that set (and, possibly, more than one subspace). The same conclusion regarding the situation *LightMalfunctions* follows from table 2, but the set of subspaces is different. And, actually, the same conclusion applies to any arbitrary situation – for any situation there exists a set of subspaces, which cover the entire context space. Within each of those subspaces the situation is linear. Those subspaces are referred to as subspaces of linearity – subspaces where the situation formula is a linear function. The proof that subspaces of linearity exist for any situation is presented in lemma 3.4.

Table 2. *LightMalfunctions* – expanded formula.

Subspace	LightMalfunctions (*SwitchPosition=On*)	LightMalfunctions (*SwitchPosition=Off*)
$\text{LightLevel} \leq 350, \text{NoiseLevel} \in (-\infty, +\infty)$	1	0.5
$\text{LightLevel} \in [350; 500]$, $\text{NoiseLevel} \in (-\infty, +\infty)$	$-\frac{1}{300}LightLevel + \frac{13}{6}$	$-\frac{1}{300}LightLevel + \frac{5}{3}$
$\text{LightLevel} \geq 500, \text{NoiseLevel} \in (-\infty, +\infty)$	0.5	0

Lemma 3.4. For any context space and any situation *Situation(X)* (defined by formula (23)) there exist a set of subspaces, with following properties:

1) Any context state is within some subspace of the set.
2) The situation is a linear function in any subspace of the set.

The input to the situation *Situation(X)* is numeric. Otherwise, operation 3.3 should be applied first.

Proof.

An arbitrary situation *Situation(X)*, compliant with the definition (23): $Situation(X) = \sum_{i=1}^{N} w_i * \mu_i(x_i) + w_0$. Membership functions are compliant with the definition (2). Let the breakpoints of membership function $\mu_i(x_i)$ be *p(i,1)…p(i,L$_i$)*, the linear coefficients be *a(i,1)..a(i,L$_i$+1)* and the bias terms be *b(i,1)..b(i,L$_i$+1)*. Let's denote the intervals in a following manner: the interval $(-\infty, p(i,1)]$ is denoted as *I(i,1)*, the interval $[p(i,1), p(i,2)]$ is denoted as *I(i,2)* and so on. The last interval of the context attribute, $[p(L), +\infty)$ is denoted as *I(i, L$_i$+1)*. Formula (30) shows membership function for i-th context attribute. The meaning does not differ from formula (3), the only difference is new notation, which will simplify further proofs.

$$\mu_i(x_i) = \begin{cases} a(i,1) * x_i + b(i,1), x_i \in I(i,1) \\ a(i,2) * x_i + b(i,2), x_i \in I(i,2) \\ \dots \\ a(i,L_i) * x_i + b(i,L_i), x_i \in I(i,L_i) \\ a(i,L_i+1) * x_i + b(i,L_i+1), x_i \in I(i,L_i+1) \end{cases} \quad (30)$$

Consider the following set of subspaces, defined by expression (31).

$$x_1 \in I(1,k_1) \wedge x_2 \in I(2,k_2) \wedge \dots \wedge x_N \in I(N,k_N) \quad (31)$$

In formula (31) every index k_i can have any integer value between 1 and L_i+1 (for i=1…N). The subspace is defined as Cartesian product of intervals over different context attributes. The formula of a situation (32) is applicable within any arbitrary subspace. Formula (32) is the result of direct substitution of (30) into the formula (23).

$$Situation(X) = \sum_{i=1}^{N} w_i * a(i,k_i) * x_i + w_0 + \sum_{i=1}^{N} w_i * b(i,k_i), \quad x_1 \in I(1,k_1) \wedge x_2 \in I(2,k_2) \wedge \dots \wedge x_N \in I(N,k_N) \quad (32)$$

The term $w_0 + \sum_{i=1}^{N} w_i * b(i,k_i)$ is a constant and the coefficients $w_i * a(i,k_i)$ are linear coefficients for i-th context attribute value x_i. Therefore, formula (32) depends linearly on all context attribute values x_i, and it proves proposition 2 of the lemma. Every formula from tables 1 and 2 are, actually, the applications of formula (32) to the situations *ConditionsAcceptable* and *LightMalfunctions* respectively.

In order to prove proposition 1, consider arbitrary context state {x_1, x_2, …, x_N}. By the construction, the set of intervals *I(1,0), I(1,1), ..., I(1,L$_1$), I(1, L$_1$+1)* covers all possible values of 1st context attribute, from -∞ to +∞. Therefore, x_1 belongs to one of those intervals. Actually, if the value x_1 it is equal to p(1,1), p(1,2), … p(1,L_1), then x_1 can belong to two intervals at once, in that case, we choose arbitrary interval. Let the interval be *I(1,k$_1$)* where k_1 can be any value between 1..L_1+1.

The same derivations can be applied to 2nd context attribute. Therefore, the value x_2 belongs to the interval *I(2,k$_2$)* where k_2 can be any value between 1 and L_2+1. And so on

until x_N. As a summary, the context state belongs to the subspace $x_1 \in I(1, k_1) \wedge x_2 \in I(2, k_2) \wedge \ldots \wedge x_N \in I(N, k_N)$, where every index k_i can have any integer value between 1 and L_i+1 (for i=1…N). So, any arbitrary context attribute belongs to some subspace, defined by (31). It proves proposition 1 of the lemma and completes the proof.
Q.E.D.■

To summarize, this section has proven that for any situation there exist a set of subspaces, where the situation is a linear function. Next sections extend this solution for the case of conjunction of several situations, and utilize it to find the maximum value of a DNF disjunct.

3.5 Subspaces of Linearity – Conjunction of Situations

As section 3.2 shows, in order to verify situations and find a counterexample, we need to find the maximum values of every disjunct of the DNF expression under verification. A disjunct in the DNF situation algebra formula is a conjunction of situations and their negations, generic example is presented in expression (33)

$$Disj(X) = Conj_1(X) \,\&\, Conj_2(X) \,\&\, \ldots \,\&\, Conj_K(X) \quad (33)$$

Combined with the situation algebra logic formulas (9), the final value is the minimum value between all the conjuncts (expression (34)).

$$\forall X \epsilon \mathbb{S}\mathbb{t}, \; Disj(X) = \min(Conj_1(X), Conj_2(X), \ldots, Conj_N(X)) \quad (34)$$

Every conjunct $Conj_i(X)$ is either a single situation, or a negation of a single situation. As section 3.3 shows, in both cases the definition is compliant with formula (23). Therefore, lemma 3.4 can be applied to every conjunct, and for every conjunct the context space can be divided into a set of subspaces, where in each subspace the conjunct is a linear function. All the conjuncts are linear functions in the intersections of the corresponding subspaces. Consider the following example.

The expression *ConditionsAcceptable(X) & LightMalfunctions(X)* should be tested for emptiness. As table 1 and table 2 show, in some subspaces of the context space the situation *ConditionsAcceptable(X)* and the situation *LightMalfunctions(X)* are linear. All the intersections of the subspaces from tables 1 and 2 are presented in table 3. The intersections are the same for the cases *SwitchPosition=On* and *SwitchPosition=Off*.

1. Any context state belongs to some subspace of the set.
2. In every subspace all the situations, mentioned in conjunction *ConditionsAcceptable(X) & LightMalfunctions(X)* are linear.

The approach can be generalized for any arbitrary conjunction. Consider a generic conjunction, defined according to formula (33). Algorithm 3.5 proposes a solution to find the mentioned subspace intersections.

Algorithm 3.5. Consider a conjunction, defined according to expression (33). The conjuncts $Conj_1$, $Conj_2$, …, $Conj_K$ each are defined according to formula (35).

The most important properties of the subspaces defined in table 3 are following:

$$Conj_K(X) = \sum_{i=1}^{N} w_i * \mu_{i,k}(x_i) + w_{0,k} \quad (35)$$

The membership function $\mu_{i,k}(x_i)$ is the function for k-th conjunct over i-th context attribute. Let the breakpoints be $p(i,k,1) \ldots p(i,k,L(i,k))$. Variable $L(i,k)$ denotes the number of breakpoints.

Table 3. Subspaces of linearity – *ConditionsAcceptable* & *LightMalfunctions*

Subspace	**ConditionsAcceptable**	**LightMalfunctions (SwitchPosition=On)**	**LightMalfunctions (SwitchPosition=Off)**
LightLevel ≤ 350 NoiseLevel ≤ 40	0.5	1	0.5
LightLevel ≤ 350 NoiseLevel ∈ [40,60]	$-\frac{1}{40}NoiseLevel+\frac{3}{2}$	1	0.5
LightLevel ≤ 350 NoiseLevel ≥ 60	0	1	0.5
LightLevel ∈ [350; 500], NoiseLevel ≤ 40	$\frac{1}{300}LightLevel-\frac{2}{3}$	$-\frac{1}{300}LightLevel+\frac{13}{6}$	$-\frac{1}{300}LightLevel+\frac{5}{3}$
LightLevel ∈ [350; 500] NoiseLevel ∈ [40,60]	$\frac{1}{300}LightLevel-$ $-\frac{1}{40}NoiseLevel+\frac{1}{3}$	$-\frac{1}{300}LightLevel+\frac{13}{6}$	$-\frac{1}{300}LightLevel+\frac{5}{3}$
LightLevel ∈ [350; 500] NoiseLevel ≥ 60	$\frac{1}{300}LightLevel-\frac{7}{6}$	$-\frac{1}{300}LightLevel+\frac{13}{6}$	$-\frac{1}{300}LightLevel+\frac{5}{3}$
LightLevel ≥ 500, NoiseLevel ≤ 40	1	0.5	0
LightLevel ≥ 500 NoiseLevel ∈ [40,60]	$-\frac{1}{40}NoiseLevel+2$	0.5	0
LightLevel ≥ 500 NoiseLevel ≥ 60	0.5	0.5	0

Algorithm pseudocode.

// **Step 1. Generate breakpoints.**
for i = 1 to N //For every context attribute
p'(i) = new BreakpointList(); //Establish a new list of breakpoints for i-th context attribute
for j = 1 to K //For every situation
p'(i).addAll (p(i,k,1…L(i,k))); //Add all breakpoints to a resulting list
end for //End for every situation
p'(i).sort();//Sort the breakpoints in ascending order
p'(i).removeDuplicates(); //All breakpoints should be distinct.
end for //End for every context attribute

//Step 2. Generate the intervals.
for i = 1 to N //For every context attribute
Interval I(i,1) = (-∞, p'(i,1)];//First interval – from negative infinity to the first breakpoint
for j = 2 to p'(i).length
Interval I(i,j) = [p'(i,j-1), p'(i,j)] //Subsequent intervals – between the breakpoints
end for
Interval I(i, p'(i).length+1) = [p'(i,L),+∞);//Last interval – from the last breakpoint and further

end for //End for every context attribute

//Step 3. Generate the subspaces.
SubspaceList resultingList = new SubspaceList();
//Generate new subspace by
for i_1 *= 1 to I(1).length // ... combining every interval over first context attribute…*
for i_2 *= 1 to I(2).length // ... with every interval of second context attribute*
...
for i_N *= 1 to I(N).length // ... with every interval of N-th context attribute*
resultingList.add (new Subspace(I(1,i_1*), I(2,*i_2*),…, I(N,*i_N*))); //add to the list*
end for
...
end for
end for

return resultingList;

Let us consider an application of the algorithm to the motivating scenario. Breakpoints over the context attribute *LightLevel* are {350,500}, the same for *ConditionsAcceptable* and *LightMalfunctions*. In the first step of the algorithm the breakpoints are merged into a single list: {350, 500, 350, 500}. The procedure *p'(i).sort()* sorts the breakpoint array and transforms it into {350, 350, 500, 500}. The procedure *p'(i).removeDuplicates()* removes duplicated breakpoints and makes the breakpoint array for *LightLevel* look like {350, 500}.

The breakpoints over the context attribute *NoiseLevel* are {40, 60} for the situation *ConditionsAcceptable*. For *LightMalfunctions* the set of breakpoints for *NoiseLevel* is empty. Therefore, the resulting set of breakpoints is {40, 60}. Removing duplicated breakpoints and sorting them does not change the array, it still remains {40, 60}.

Step 2 of the algorithm composes the intervals. For the context attribute *LightLevel* the intervals are (-∞, 350], [350,500] and [500, +∞). For the context attribute *NoiseLevel* the intervals are (-∞, 40], [40,60] and [60, +∞).

Step 3 composes the subspaces out of intervals, obtained in the step 2. The generated subspaces are illustrated in table 3. Also table 3 shows that all the mentioned situations are linear inside those subspaces. The proof of the algorithm shows that for an arbitrary conjunction all the conjuncts are linear inside the subspaces, generated by the algorithm 3.5.

Proof. In order to prove the correctness of the algorithm, it is sufficient to prove that the algorithm complies with two properties.

1. Any context state is inside at least one subspace of the set.
2. In every subspace all the situations within the conjunction are linear.

Property 1 can be proven as follows, it slightly resembles the proof of lemma 3.4. Consider an arbitrary context state $\{x_1, x_2, \ldots, x_N\}$. Consider the value of 1st context attribute x_1. By the construction (see step 2), the set of intervals *I(1,1), …, I(1,I(1).length)* covers all possible values of 1st context attribute, from -∞ to +∞. So, x_1 belongs to at least one of those intervals. Let it be the interval $I(1,k_1)$. The same logic can be applied to any arbitrary context attribute: the context attribute value x_i belong to the interval $I(i,k_i)$. Therefore, context state $\{x_1, x_2, \ldots, x_N\}$ belongs to the subspace $x_1 \in I(1,k_1) \wedge x_2 \in I(2,k_2) \wedge \ldots \wedge x_N \in I(1,k_N)$. According to the step 2 of the algorithm, that subspace was generated in the nested loops where $i_1=k_1$, $i_2=k_2$, …, $i_N=k_N$. Therefore, an arbitrary context state belongs to one of the generated subspaces, Q.E.D. for property 1.

Property 2 can be proven as follows. According to lemma 3.4, for any situation there exist a set of subspaces, where the situation is linear for every subspace in the set. As a consequence, the situation is linear in every subspace of that subspace. We need to prove that any subspace, generated by the algorithm, is a subspace of a subspace of linearity for any conjunct. Consider the proof of a following hypothesis.

Hypothesis 3.5.1. Any interval over any context attribute, generated in the step 2 of the algorithm, does not contain any breakpoints (except for on the boundaries) of any conjuncts of that context attributes. Consider a proof by contradiction. Let there be some generated subspace $x_1 \in I(1,k_1) \wedge x_2 \in I(2,k_2) \wedge \ldots \wedge x_N \in I(1,k_N)$. Let p_i be the breakpoint of conjunct $Conj_t(X)$ on i-th context attribute. Let p_i be inside the interval $I(i,k_i)$ and not on its boundaries. However, according to the step 2 of the algorithm any breakpoint is a boundary of some interval - the breakpoint p_i should have been added as a breakpoint in the step 1 and could not have been removed by *removeDuplicates()* operation (if it is a duplicate, one of the p_i points is preserved to become a boundary of the interval in the step 2). Therefore, the point p_i should have become a boundary of some interval, let it be the interval $I(i,m)$. According to the assumption of proof by contradiction the point p_i belongs to the interval $I(i,k_i)$, therefore, the intervals $I(i,m)$ and $I(i,k_i)$ do overlap. Also according to the assumption of proof by contradiction the point p_i is not on the boundary of $I(i,k_i)$, therefore the intersection of $I(i,m)$ and $I(i,k_i)$ is not restricted to the boundary point. However, by the construction in the step 2 of the algorithm, the intervals can intersect only on boundary points. It is a contradiction, therefore, such breakpoint p_i does not exist. **Q.E.D. for hypothesis 3.5.1.**

Consider a proof of proposition 2 for an arbitrary conjunct $Conj_t(X)$ and an arbitrary generated subspace $x_1 \in I(1,k_1) \wedge x_2 \in I(2,k_2) \wedge \ldots \wedge x_N \in I(1,k_N)$. Consider an arbitrary i-th context attribute and the interval $I(i,k_i)$. According to the hypothesis 3.5.1, membership function of $Conj_t(X)$ over the first context attribute contains no breakpoints inside the interval $I(i,k_i)$ (except for, possibly, on the boundaries). There can be three cases:

1. The interval $I(i,k_i)$ is situated before the first breakpoint $p(i,t,1)$, including the case when the first breakpoint is an upper boundary of the interval $I(i,k_i)$. In this case the interval $I(i,k_i)$ belongs to the interval $(-\infty;p(i,t,1)]$, where $Conj_t(X)$ linearly depends on i-th context attribute value according to definitions (23) and (2). Therefore, inside the interval $I(i,k_i)$ the conjunct $Conj_t(X)$ linearly depends on i-th context attribute value.

2. The interval $I(i,k_i)$ is after the last breakpoint, including the case when the last breakpoint is on the lower boundary of the interval $I(i,k_i)$. If $Conj_t(X)$ contains no breakpoints for i-th context attribute, it can be viewed as a special case of this point or previous point. In this case the interval $I(i,k_i)$ belongs to the interval $[p(i,t,L(i,t)); \infty)$, where $Conj_t(X)$ linearly depends on i-th context attribute value. As well as for case 1, inside the interval $I(i,k_i)$ the conjunct $Conj_t(X)$ linearly depends on i-th context attribute value due to definitions (23) and (2).

3. The interval $I(i,k_i)$ is between the breakpoints and, hence, inside the interval $[p(i,t,m); p(i,t,m+1)]$. But inside $[p(i,t,m); p(i,t,m+1)]$ the conjunct $Conj_t(X)$ linearly depends on i-th context attribute value, and the same applies to the subinterval $I(i,k_i)$.

The remaining options are ruled out by the hypothesis 3.5.1. If the interval does not contain any breakpoint, then it is either before the first breakpoint, after the last breakpoint or between the breakpoint.

Therefore, in the interval $I(i,k_i)$ the conjunct $Conj_t(X)$ linearly depends on i-th context attribute value. By applying the same proof to all conjuncts and all context attributes, we can state that in the subspace $x_1 \in I(1,k_1) \wedge x_2 \in I(2,k_2) \wedge \ldots \wedge x_N \in I(1,k_N)$ all the conjuncts linearly depend on all context attribute values. And it can be applied to any subspace,

generated by the algorithm. As a summary, for all subspaces generated by the algorithm all the conjuncts linearly depend on all context attribute values. It completes the proof of proposition 2 and, hence, completes the proof of algorithm 3.5.

Q.E.D.■

The next section discusses searching for the maximum point of a conjunction within every subspace, generated by the algorithm 3.5. Global maximum can be obtained straightforwardly by comparing the maxima within the subspaces, and it completes emptiness test of a DNF situation algebra expression.

3.6 Constrained Optimization in the Subspace

Previous section described the algorithm to divide a context space into a set of subspaces with special properties. In the subspaces, generated by the algorithm 3.5, all the conjuncts are linear functions. The task of finding a maximum of a conjunction inside the subspace can be reduced to the linear programming task. Consider an example from motivating scenario, before we proceed to the proof for an arbitrary conjunction.

In the motivating scenario the conjunction under testing is *ConditionsAcceptable(X) & LightMalfunctions(X)*. Context space can be divided into the set of subspaces, where both *ConditionsAcceptable(X)* and *LightMalfunctions(X)* are linear. The subspaces, as well as linear functions, are presented in table 3. For illustration, let's find maximum value within the subspace $(\text{LightLevel} \in [350; 500])\text{^}(\text{NoiseLevel} \in [40,60])$ in case when *SwitchPosition=On*. Expression (36) formalizes the maximization task.

$$\begin{aligned}&\text{Maximize: } min\left(\tfrac{1}{300}LightLevel - \tfrac{1}{40}NoiseLevel + \tfrac{1}{3}, -\tfrac{1}{300}LightLevel + \tfrac{13}{6}\right)\\&\text{w.r.t. constraints:}\\&\quad \text{LightLevel} \le 500\\&\quad \text{LightLevel} \ge 350\\&\quad \text{NoiseLevel} \ge 40\\&\quad \text{NoiseLevel} \le 60\end{aligned} \tag{36}$$

Actually, the task (36) is a piecewise linear programming task [BV04], which can be reduced to the linear programming tasks. Task (36) can be rewritten as task (37) by introducing the artificial intermediate variable $t=min\left(\tfrac{1}{300}LightLevel - \tfrac{1}{40}NoiseLevel + \tfrac{1}{3}, -\tfrac{1}{300}LightLevel + \tfrac{13}{6}\right)$.

$$\begin{aligned}&\text{Maximize: } t\\&\text{w.r.t. constraints:}\\&\quad t \le \frac{1}{300}LightLevel - \frac{1}{40}NoiseLevel + \frac{1}{3}\\&\quad t \le -\tfrac{1}{300}LightLevel + \tfrac{13}{6}\\&\quad 350 \le \text{LightLevel} \le 500\\&\quad 40 \le \text{NoiseLevel} \le 60\end{aligned} \tag{37}$$

New maximization objective is the result of direct substitution of new variable *t*. The two uppermost constraints directly follow from the definition of *t*. Task (37) can be further rewritten as task (38).

Maximize: t
w.r.t. constraints:

$$\begin{aligned} & t - \frac{1}{300} LightLevel + \frac{1}{40} NoiseLevel \leq \frac{1}{3} \\ & t + \tfrac{1}{300} LightLevel \leq \tfrac{13}{6} \\ & 350 \leq \text{LightLevel} \leq 500 \\ & 40 \leq \text{NoiseLevel} \leq 60 \end{aligned} \tag{38}$$

The task (38) is not yet a canonical form of the linear programming task. However, for many linear programming solvers the format (38) is already acceptable as an input. For this article we use the implementation of interior point method, embedded in the GNU Linear Programming Kit [M12], which in turn is embedded in GNU Octave [E08]. The maximum values, obtained by solving of piecewise linear maximization task for different subspaces, generated by algorithm 3.5, and different subtasks, generated by operation 3.3, are presented in table 4.

Now the verification of motivating scenario is complete. Rows 4 and 5 of table 4 show that for the values *LightLevel*=425, *NoiseLevel*=40 and *SwitchPosition=On* the confidence level for the expression *ConditionsAcceptable(X) & LightMalfunctions(X)* is equal to 0.75. It is above the threshold of 0.7, therefore, for the expression the *ConditionsAcceptable(X)&LightMalfunctions(X)* verification has detected an error. A straightforward test can show, that it is really a counterexample: for the context state {425; 40; On} certainty of *ConditionsAcceptable(X)* and certainty of *LightMalfunctions(X)* will be both 0.75. Both situations will be triggered in that case, and it will cause inconsistent situation awareness results.

Table 4 shows that the counterexample is detected twice (rows 5 and 6). The reason is that the counterexample lies on the border of the subspaces. It is quite common case and it does not hamper verification in any way. The found counterexample is not the only counterexample available – within some subspace around the counterexample the confidence value of *ConditionsAcceptable(X) & LightMalfunctions(X)* is above 0.7.

Consider a sequence of steps for any arbitrary conjunction. Let the conjunction be $Conj_1(X)$ &$Conj_2(X)$ &…&$Conj_K(X)$. Let the subspace be ($low_1 \leq x_1 \leq high_1$) ^($low_2 \leq x_2 \leq high_2$)^…($low_N \leq x_N \leq high_N$). Any lower or upper boundary can as well be infinite, in that case inequality sign is not inclusive. The subspace is generated by the algorithm 3.5 for the conjunction. Therefore, every conjunct is linear inside the subspace. Let's denote the linear coefficients of an arbitrary l-th conjunct according to formula (39).

$$Conj_l(X) = \sum_{i=1}^{N} a(l,i) * x_i + b(l) \tag{39}$$

The maximization task looks as follows (formula (40)).

$$\begin{aligned} \text{Maximize: } & min\left(\sum_{i=1}^{N} a(1,i) * x_i + b(1), \sum_{i=1}^{N} a(2,i) * x_i + b(2), \right. \\ & \left. \dots, \sum_{i=1}^{N} a(K,i) * x_i + b(K)\right) \\ \text{w.r.t. constraints:} & \\ & low_1 \leq x_1 \leq high_1 \\ & low_2 \leq x_2 \leq high_2 \\ & \dots \\ & low_N \leq x_N \leq high_N \end{aligned} \tag{40}$$

Table 4. *ConditionsAcceptable(X)&LightMalfunctions(X)* - Maxima within subspaces.

№	Subspace	Maximum (t-value)	Context State [LightLevel; NoiseLevel]
1	$\text{LightLevel} \leq 350, \text{NoiseLevel} \leq 40$, SwitchPosition=On	0.5	[350; 40]
2	$\text{LightLevel} \leq 350, \text{NoiseLevel} \in [40,60]$, SwitchPosition=On	0.5	[350; 40]
3	$\text{LightLevel} \leq 350, \text{NoiseLevel} \geq 60$, SwitchPosition=On	0	[350; 60]
4	$\text{LightLevel} \in [350; 500], \text{NoiseLevel} \leq 40$, SwitchPosition=On	0.75	[425; 40]
5	$\text{LightLevel} \in [350; 500], \text{NoiseLevel} \in [40,60]$, SwitchPosition=On	0.75	[425; 40]
6	$\text{LightLevel} \in [350; 500], \text{NoiseLevel} \geq 60$, SwitchPosition=On	0.5	[500; 60]
7	$\text{LightLevel} \geq 500, \text{NoiseLevel} \leq 40$, SwitchPosition=On	0.5	[500; 40]
8	$\text{LightLevel} \geq 500, \text{NoiseLevel} \in [40,60]$, SwitchPosition=On	0.5	[500; 40]
9	$\text{LightLevel} \geq 500, \text{NoiseLevel} \geq 60$, SwitchPosition=On	0.5	[500; 60]
10	$\text{LightLevel} \leq 350, \text{NoiseLevel} \leq 40$, SwitchPosition=Off	0.5	[350; 40]
11	$\text{LightLevel} \leq 350, \text{NoiseLevel} \in [40,60]$, SwitchPosition=Off	0.5	[350; 40]
12	$\text{LightLevel} \leq 350, \text{NoiseLevel} \geq 60$, SwitchPosition=Off	0	[350; 60]
13	$\text{LightLevel} \in [350; 500], \text{NoiseLevel} \leq 40$, SwitchPosition=Off	0.5	[350; 40]
14	$\text{LightLevel} \in [350; 500], \text{NoiseLevel} \in [40,60]$, SwitchPosition=Off	0.5	[350; 40]
15	$\text{LightLevel} \in [350; 500], \text{NoiseLevel} \geq 60$, SwitchPosition=Off	0.25	[425; 60]
16	$\text{LightLevel} \geq 500, \text{NoiseLevel} \leq 40$, SwitchPosition=Off	0	[500; 40]
17	$\text{LightLevel} \geq 500, \text{NoiseLevel} \in [40,60]$, SwitchPosition=Off	0	[500; 40]
18	$\text{LightLevel} \geq 500, \text{NoiseLevel} \geq 60$, SwitchPosition=Off	0	[500; 60]

Task (40) is a piecewise linear programming task. Piecewise linear programming task can be reduced to linear programming task by introduction of slack variable [BV04]. The result looks as follows (task (41)).

Maximize: t

w.r.t. constraints:

$$
\begin{aligned}
& t + \sum_{i=1}^{N}(-a(1,i)) * x_i \leq b(1) \\
& t + \sum_{i=1}^{N}(-a(2,i)) * x_i \leq b(2) \\
& \ldots \\
& t + \sum_{i=1}^{N}(-a(K,i)) * x_i \leq b(K) \\
& low_1 \leq x_1 \leq high_1 \\
& low_2 \leq x_2 \leq high_2 \\
& \quad \ldots \\
& low_N \leq x_N \leq high_N
\end{aligned}
\tag{41}
$$

Task (41) can be further reduced to the canonical form of linear programming task (see [BV04]). However, linear programming solvers often can work with the task in (41) format

already. As a result of solving task (41), the variable t will contain the maximum confidence value of the conjunction $min\ (\sum_{i=1}^{N} a(1,i) * x_i + b(1),\ \sum_{i=1}^{N} a(2,i) * x_i + b(2),\ \ldots,\ \sum_{i=1}^{N} a(K,i) * x_i + b(K))$, which should be tested against the threshold. The solution of task (41) in the values $x_1 \ldots x_N$ will also contain a set of context attribute values, for which the maximum certainty is achieved. If the maximum certainty exceeds the threshold, this set of values is a counterexample.

This section completes the description and proof of the verification algorithm. Next section summarizes the entire verification procedure.

3.7 Verification Approach – Summary

Summarizing the information of sections 3.1-3.6, the final verification procedure looks as follows.

Input. The property under verification should be represented as emptiness assertion for DNF situation algebra expression. The guidelines for composing the expression are presented in section 2.3 and section 3.2.

Step 1. Use operation 3.3 to define the subtasks for every combination of non-numeric and mixed context attributes.

Step 2. For every subtask and for every disjunct of the DNF expression, find the subspaces where all the conjuncts are linear. It can be done using the algorithm 3.5.

Step 3. For every subtask, for every disjunct of the expression and for every subspace, identified in the step 2, define and solve linear programming task to find the maximum certainty. The procedure for defining and solving linear programming task is defined in section 3.6.

Linear programming solution contains maximum certainty value, as well as the context state where the maximum is achieved. If the certainty is above the threshold, then the corresponding context state is a counterexample. Counterexample is added to the list. If the developer needs just a Boolean answer whether the verification found any errors, then verification can stop upon finding the first counterexample. If after all iterations of step 3 no counterexamples are found, it means that the context model complies with the assertion, and the verification detected no errors.

Output. The list of counterexamples – context states where the expression under emptiness exceeds the threshold. If the list is empty, then verification has found no errors.

The verification algorithm can be described in a following pseudocode.

Algorithm 3.7.

```
// Step 1. Define subtasks (use operation 3.3).
subtasks = defineSubtasks(expression, situations);

//Step 2. For every subtask and for every disjunct of DNF expression, find the subspaces of linearity.
for i = 1 to expression.getDisjuncts()
  for j = 1 to subtasks.getCount()
    //Use algorithm 3.5 to get subspaces of linearity
    subspaces[i,j] = subtasks[i].getSubspaces(expression.disjunct[j]);
  end for; // End for every subtask
end for; // End for every disjunct
```

```
//Step 3. For every subspace of linearity – find maximum confidence value.
for i = 1 to expression.getDisjuncts()
  for j = 1 to subtasks.getCount()
     for k = 1 to subspaces[i,j].length //For every subspace
       //Define linear programming subtask according to guidelines (41)
       linearProgrammingTask = defineLPTask(subspaces[i,j,k], expression.disjunct[i], subtask[j]);
       //Solve linear programming task, find maximum confidence value and corresponding context state
       [value contextState] = solveLinearProgrammingTask(linearProgrammingTask);
       if (value >= threshold)
          counterexamples.add(contextState, expression.getConfidence(contextState));
       end if
    end for; // End for every subspace
  end for; // End for every subtask
end for; //End for every disjunct

//Output: a list of counterexamples. An empty list means that verification found no errors.
return counterexamples;
```

As for the motivating scenario, the proposed algorithm found a counterexample {*LightLevel*=425, *NoiseLevel*=40, *SwitchPosition=On}*, for which both situations have certainty of 0.75. The verification has detected an error.

The next section discusses theoretical and practical complexity of the proposed solution.

4 Evaluation

In section 3 we proposed, discussed and proved the sequence of the steps of the verification approach for fuzzy situation models in pervasive computing. Complexity of the steps of the verification algorithm consists of following components:

- Generation of possible combinations of non-numeric parameters.
- Generation of subspaces.
- Defining and solving linear programming task for every subspace and every combination.

The verification mechanisms were implemented as an extension over ECSTRA (Enhanced Context Spaces Theory-based Reasoning Architecture) situation awareness framework, and all the practical tests were performed inside that framework as well. For more details on ECSTRA refer to the paper [BZ11a].

Next sections discuss the components of complexity in more details. Section 4.1 and 4.2 prove the complexity of subspace generation and working with linear programming tasks respectively. Section 4.3 generalizes the result for non-numeric and mixed context attributes. Section 4.4 provides the summary and finalizes complexity estimation.

4.1 Complexity Analysis for Generation of Subspaces

Algorithm 3.5 is responsible for decomposition of context space into the subspaces. The algorithm is prone to multiple ways of enhancement, which we leave for the future work. Here we analyze the complexity of the algorithm 3.5 in a way it is presented in section 3.5.

The notation is as follows. Consider that algorithm 3.5 has an input like expression (33) – a conjunction containing K conjuncts, which are defined over N context attributes. Let's denote the number of breakpoints over i-th context attribute for j-th conjunct as *L(i,j).* Assume that the input is correct and all the conjuncts are defined properly, without duplicate breakpoints.

Step 1 of the algorithm 3.5 involves merging the breakpoints, sorting them and removing the duplicates. Consider the complexity of that operation for an arbitrary context attribute. Let's denote it as i-th context attribute, where *i* can be any integer from 1 to N. Sorting can be efficiently done, for example, by Quicksort algorithm [Ho62] in, effectively, O(T*logT) time, where T is the number of elements in an array. The number of elements in array is, actually, the number of breakpoints before removing the duplicates, which is the sum of all breakpoints of all the conjuncts. Therefore, the complexity is $O((\sum_{j=1}^{K} L(i,j))*\log(\sum_{j=1}^{K} L(i,j)))$. In the worst case Quicksort can reach $O(T^2)$ time.

In a sorted array the duplicates can be found in $O(\sum_{j=1}^{K} L(i,j))$ time with a trivial algorithm. So, out of two sequential substeps of step 1 of algorithm 3.5, sorting substep has polylogarithmic complexity (quadratic in the worst case) and the substep of removing duplicates has linear complexity. The total complexity for a single context attribute is, therefore, polylogarithmic $O((\sum_{j=1}^{K} L(i,j))*\log(\sum_{j=1}^{K} L(i,j)))$ in average case or quadratic $O\left(\left(\sum_{j=1}^{K} L(i,j)\right)^2\right)$ in the worst case. For all context attributes the complexity is illustrated in expression (42).

$$\text{Average case complexity: } O(\sum_{i=1}^{N}[\ (\sum_{j=1}^{K} L(i,j)) * \log(\sum_{j=1}^{K} L(i,j))\ ]) \quad (42)$$
$$\text{Worst case complexity: } O(\sum_{i=1}^{N}[\ (\sum_{j=1}^{K} L(i,j))^2\ ])$$

Complexity of the steps 2 and 3 of the algorithm 3.5 depends on the final number of breakpoints. That number can be estimated as follows. The total number of breakpoints over i-th context attribute we denote as *L(i).* By the construction of algorithm 3.5, every breakpoint of every conjunct becomes a breakpoint for defining subspaces. The operation *removeDuplicates()* then removes duplicate values, which might appear from different conjuncts, and does not change previously mentioned fact – if a breakpoint appeared in a conjunct, it will appear in the final set of breakpoints. Therefore, total number of breakpoints cannot be smaller than the number of breakpoints for any of the conjuncts, and we can estimate that the total number of breakpoints $L(i) \geq max_j(L(i,j))$. The lower boundary is achieved in motivating scenario for both context attributes. For context attribute *LightLevel* there are two breakpoints both for *ConditionsAcceptable(X)* and for *LightMalfunctions(X)* (350 Lx and 500 Lx in both cases). Therefore, it can't be less than two breakpoints in final breakpoints set, and those breakpoints are 350 Lx and 500 Lx. For the context attribute *NoiseLevel* the number of breakpoints also reach lower boundary – there are zero breakpoints for *LightMalfunctions(X)* and two for *ConditionsAcceptable(X)* (30 dB and 60 dB). The final number of breakpoints is two – those are 30 dB and 60 dB.

The largest number of breakpoints will be generated in the case when there are no duplicates (i.e. *removeDuplicates()* does not change anything). In that case the number of breakpoints for i-th context attribute is $\sum_{j=1}^{K} L(i,j)$. In motivating scenario in some way it happens for the context attribute *NoiseLevel* (lower and upper boundaries are the same for that case). The situation *ConditionsAcceptable(X)* has two breakpoints over *NoiseLevel* (30 dB and 50 dB) and the situation *LightMalfunctions(X)* has zero breakpoints over that context attribute. The total number of breakpoints is two.

To summarize, for i-th context attribute the number of breakpoints can be estimated as expression (43).

$$max_j(L(i,j)) \leq L(i) \leq \sum_{j=1}^{K} L(i,j) \quad (43)$$

Step 2 of the algorithm 3.5 generates intervals from the breakpoints. Generation of an interval takes constant time, so the time complexity is proportional to the number of iterations in the nested loops: $O(\sum_{i=1}^{N} L(i))$, where according to (43), $\sum_{i=1}^{N} L(i) \leq \sum_{i=1}^{N} \sum_{j=1}^{K} L(i,j)$. Therefore, the complexity of step 2 in the worst case is $O(\sum_{i=1}^{N} \sum_{j=1}^{K} L(i,j))$, which is of lower order comparing to complexity of step 1 (both average and worst cases of expression (42)). The order of total complexity of steps 1 and 2 is determined by complexity of step 1 and is still represented by expression (42).

Step 3 of the algorithm 3.5 generates all the subspaces by calculating Cartesian product of intervals between the breakpoints, which were identified in the previous step. Once again, generating the subspace is a constant time operation. Therefore, the time complexity is determined by the number of iterations of the nested loops. The number of iterations is $\prod_{i=1}^{N}(L(i)+1)$, which in the worst case is $\prod_{i=1}^{N}(1+\sum_{j=1}^{K} L(i,j))$ (new summand +1 appears because the number of intervals is equal to the number of breakpoints plus one). This value is of higher order comparing to the total complexity of first two steps, presented in expression (42). The order of this term determines total complexity of the algorithm 3.5. To summarize, the time complexity of the algorithm 3.5 is presented in expression (44).

$$\begin{aligned} &\text{Complexity: } O(\textstyle\prod_{i=1}^{N}(L(i)+1)) = O(S) \\ &\text{Worst case complexity: } O(\textstyle\prod_{i=1}^{N}(1+\sum_{j=1}^{K} L(i,j))) \\ &\text{Best case complexity: } O(\textstyle\prod_{i=1}^{N} max_j(L(i,j)+1) \end{aligned} \quad (44)$$

In the expression (44) we denoted $\prod_{i=1}^{N}(L(i)+1)$ as S, which means the number of subspaces. Actually, the number of iterations of the nested loops of step 3 in algorithm 3.5 is the number of generated subspaces. The time complexity of the algorithm 3.5 has linear dependency on the number of subspaces, and best and worst cases of expression (44) put boundaries on the number of subspaces depending on the number of breakpoints.

The summary of expression (44) is that running time of the algorithm 3.5 linearly depends on the number of subspaces. We practically proved the time complexity of the algorithm using the following testing procedure. The testing involved 1000 experiments, where every experiment was conducted as follows.

1. Generate K random situations, which contain a random subset of N context attributes. K and N are random and vary from 1 to 6. Situations follow the definition (23): $Sit(X) = \sum_{i=1}^{N} w_i * \mu_i(x_i) + w_0$. The weights, the bias term and the breakpoints of membership functions are generated randomly in realistic manner. The shape of every membership function $\mu_i(x_i)$ is uniformly randomly chosen among the options presented in figure 2.

2. The generated K situations form a conjunction. Algorithm 3.5 is applied to detect the subspaces of linearity. The time, taken by the algorithm 3.5 is an outcome of the experiment.

We summarized the outcomes of all 1000 experiments in the figure 5 and processed the results using R toolkit [VS12]. The plot in figure 5 was generated by R toolkit as well; minor manual interventions were restricted to improving readability of the figure.

The points in figure 5 exhibit clear linear dependency. The coefficient R^2 between the time, required to generate subspaces, and the number of subspaces is 0.9995, and it practically proves the linearity. These results completely confirm our theoretical estimations of the algorithm complexity, summarized in expression (44).

It should be noted that it is enough to generate the breakpoints (step 1) and intervals (step 2) once for all the involved context attributes (including mixed). Later the same breakpoints can be used when applying algorithm 3.5 to another disjunct or another combination of non-numeric values. Subspaces can be also generated only once, if there are no mixed context attributes involved (and significantly reused if there are any). However, solving linear programming task should be done separately for every subspace and for every combination of non-numeric/mixed context attributes anyway, and, as will be proven further, it is the main source of complexity. So, those obvious enhancements do not reduce the order of complexity and, therefore, can be omitted from complexity analysis.

In order to reduce memory requirements it is possible to generate the subspaces and for every generated subspace right away calculate the maximum value of every disjunct in the DNF expression under test.

Next section discusses the verification steps after the algorithm 3.5 – defining and solving piecewise linear programming tasks.

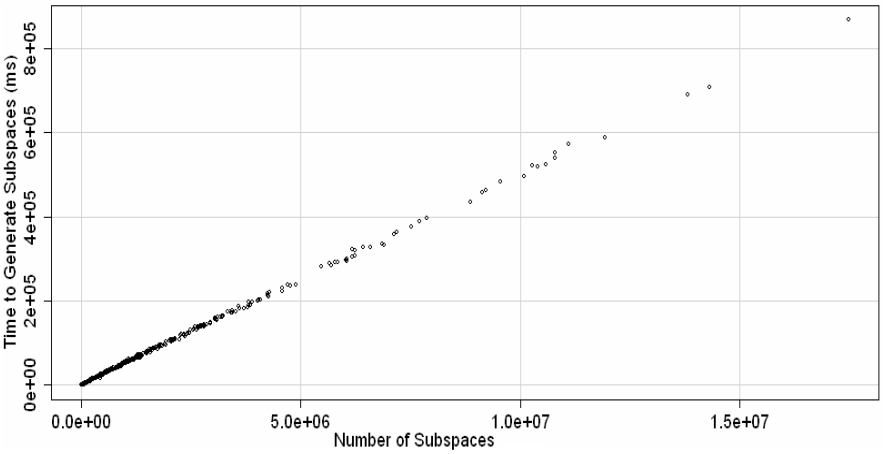


Fig. 5. Time required to generate subspaces of linearity.

4.2 Complexity Analysis of Defining and Solving Linear Programming Task

Next step of the verification algorithm involves defining and solving linear programming task for every subspace, derived in previous section. Consider complexity analysis for a single subspace, later it will be expanded for the case of all subspaces.

The definition of linear programming task requires constructing the description in the format (41). There are 2*N inequalities that emerge from subspace definition (expressions like $low_i \le x_i \le high_i$ count as two inequalities), where N is the number of context attributes. There are also K inequalities (equal to the number of conjuncts) that result from transformation of piecewise linear minimization task into linear programming task, and each of those inequalities contain N+1 terms (the coefficients for $x_1 \ldots x_N$ and t). The

construction of latter inequalities is the most computationally heavy operation (order of N*K, opposed to order of N in the previous step), and it determines the order of complexity for the whole task construction algorithm. To summarize, straightforward construction of linear programming task requires O(N*K) time, and it should be done for every subspace.

The standard form of linear programming task is defined in expression (45) [BV04].

Maximize: C^T*X
w.r.t. constraints: (45)
$A*X = b$
$X \geq 0$

In the task (45) C and b are constant vectors and A is a matrix. Conversion of the task from format (41) into format (45) requires several steps. The inequalities $low_i \leq x_i$ can be transformed into $x'_i \geq 0$ by introducing the variable $x'_i = x_i - low_i$. We also need to introduce new variables for all the other N+K inequalities in order to transform them into equalities. The main idea is to transform an inequality $\sum a_i * x_i \leq b$ into an equality $\sum a_i * x_i + x_0 = b$, which becomes a part of $A*X = b$, and inequality $x_0 \geq 0$, which becomes a part of $X \geq 0$ constraints (see [BV04] for more details). The newly introduced variables like x_0 are referred to as slack variables. Some linear programming solvers, including GLPK [M12], can handle the transformation automatically. The very process of transformation does not add much complexity, but the fact that the number of variables changes can be important for complexity analysis. In order to transform task (41) into task (45) N slack variables need to be introduced for inequalities like $t + \sum_{i=1}^{N}(-a(j,i)) * x_i \leq (j)$, and K slack variables need to be introduced for inequalities like $x_i \leq high_i$. Together with N initial variables, it makes the total number of variables 2*N + K.

A linear programming task can be solved by polynomial complexity algorithms. For example, Karmarkar's algorithm [K84] estimates the complexity of the method as $O(n^{3.5}*B)$, where n is the number of variables (after introducing slack variables), and B is the number of bits in the input. Many more polynomial algorithms can solve linear programming tasks (see [BV04][W97] for more information).

ECSTRA extension, which implements fuzzy situation verification approach, employs GLPK [M12] toolkit and its implementation of interior point method. According to GPLK reference manual [M12], the toolkit uses a version of primal-dual interior point methods, implemented according to Mehrotra's technique [M92].

In order to estimate practical complexity, we performed a set of 1000 experiments, where every experiment was conducted as follows:

1. K random situations are generated over a random subset of N context attributes. K and N were random values uniformly chosen from 1 to 50. Situations are defined according to expression (23): $Sit(X) = \sum_{i=1}^{N} w_i * \mu_i(x_i) + w_0$. The weights and the bias term are generated randomly in a realistic manner. The shape of membership functions $\mu_i(x_i)$ is randomly chosen among several options presented in figure 2. The breakpoints of membership functions are chosen randomly.

2. The situations, generated in the step 1, are merged in a conjunction. The subspaces of linearity are obtained using the algorithm 3.5.

3. A single random subspace of linearity is uniformly chosen from the set, obtained in the step 2.

4. For the subspace, chosen in the step 3, linear programming task was constructed and solved. Solution time is the outcome of the experiment. In order to mitigate possible

random disturbances, calculations for every experiment were conducted 10 times and averaged.

We summarized the outcomes of the experiment in the figure 6. The plots were created using R toolkit [VS12]; minor manual changes to the plots were restricted to improving the readability.

The main purpose of the experiment was to find out how well does the linear programming solver algorithm scale with the growing size of the task. Figure 6 shows the following results. The plot on figure 6a depicts dependency of linear programming solution time on the number of context attributes involved in subspace definition. The number of variables in linear programming task is equal to the number of context attributes plus one (variable *t*). The number of variables tends to be close to the upper boundary. When there are multiple situations, each defined over a random subset of context attributes, the number of context attributes mentioned at least once grows fast. And the subspaces have to account for every context attribute, which is mentioned in at least one situation.

Figure 6b depicts dependency between the number of constraints (i.e. number of rows of matrix *A* in the standard format (45)) and the time to solve linear programming task. The number of constraints is N+K: there are K constraints of type $t + \sum_{i=1}^{N}(-a(j,i)) * x_i \leq b(j)$ and N constraints of type $x_i \leq high_i$. Inequalities are later transformed into equalities by introduction of slack variables, but the number of equalities still remains N+K.The lower boundaries of the variables constitute $X \geq 0$ restriction of task (45), and are not counted. The dependency on figures 6a and 6b is not straightforward, but the results of experiments are fully explained by the subsequent plots 6c and 6d.

The plot on figure 6c illustrates the dependency between the number of conjuncts K and the time to solve linear programming task. The coefficient R^2 is equal to 0.9142, and it practically shows linear dependency with some variance. However, it turns that the results are better explained by the subsequent plot 6d.

The Y-axis of plot 6d, as in all plots of figure 6, is the time required to solve linear programming task. The X-axis of plot 6d is the multiplication of number of conjuncts K and number of involved context attributes N. This value corresponds to the size of the input required to define the task. In order to specify the linear programming task the following information is required. Each of K inequalities $t + \sum_{i=1}^{N}(-a(j,i)) * x_i \leq b(j)$ contains N+1 coefficients on the left side and one term *b(j)*. The boundaries of the variables require 2*(N+1) more values (infinite boundaries on *t* are implied in formula (45), but should be supplied explicitly to linear programming solver). The maximization objective requires N+1 more values for specification (the function C^TX to maximize is just *t*, so C = [1 0 0…0]). It makes the number of parameters to specify equal to (N+2)*K+2*(N+1)+(N+1). Those parameters are necessary (i.e. no parameter can be deduced from the values of other parameters) and sufficient (i.e. they define linear programming task unambiguously). The number of parameters is the value of order O(N*K), and N*K is the X-axis of the plot 6d. The coefficient R^2 is equal to 0.9148, and it practically shows even more pronounced linear dependency comparing to the figure 6c. The dependency 6c, can be explained by the fact that the number of involved context attributes tend to be close to maximum value (this effect was already described when discussing plot 6a), therefore N*K is close (and often equal) to N_{MAX}*K, where N_{MAX} is the total number of available context attributes (50 for the conducted experiments).

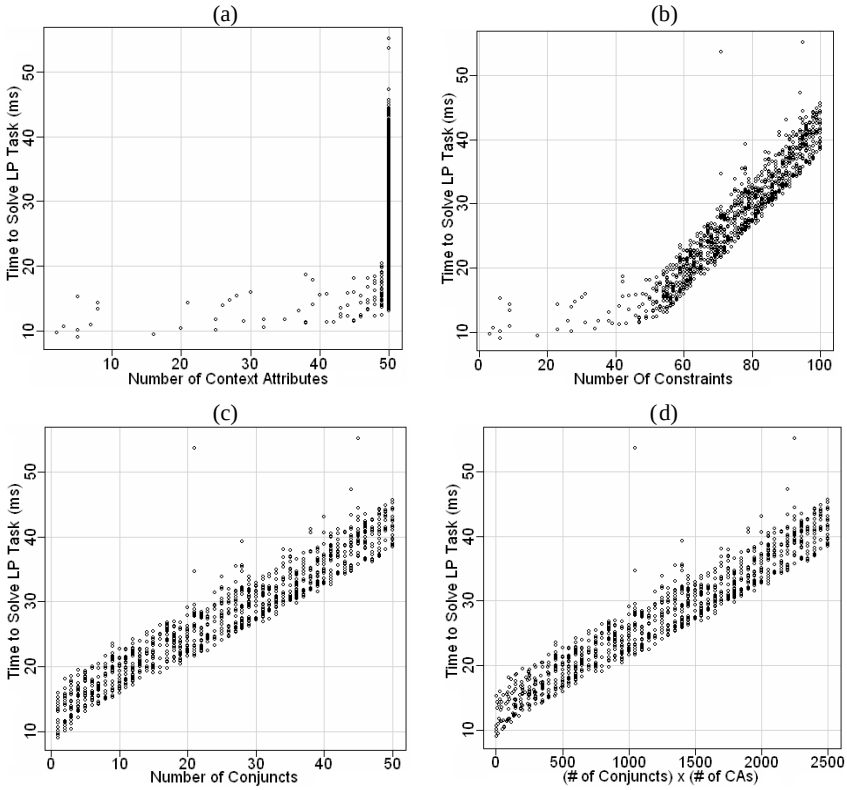


Fig. 6. Time to solve linear programming task, depending on various factors: (a) Depending on number of involved context attributes; (b) Depending on number of inequality constraints; (c) Depending on number of conjuncts; (d) Depending on multiplication of number of conjuncts and number of context attributes.

To summarize, the algorithm for defining linear programming task requires O(N*K) time. The testing has shown, that practical complexity of solving linear programming task also have the complexity of order O(N*K). Therefore, the total complexity of defining and solving LP task can be estimated as O(N*K). Combining with, estimations (43) and (44), the total complexity can be estimated as expression (46).

Complexity: O(S*N*K) (46)
$\prod_{i=1}^{N} max_j(L(i,j)+1) \leq S \leq \prod_{i=1}^{N}(1+\sum_{j=1}^{K} L(i,j))$

Expression (46) defines the verification complexity for a single conjunction with K conjuncts, defined over N context attributes that result in S subspaces of linearity. Moreover, the complexity estimate (46) implies that the verification continues until the maximums for all subspaces are found. This approach is advisable if the goal is to collect as

many counterexamples as possible. The identified counterexamples help to narrow down and fix one or multiple situation modeling problems. However, sometimes the developer might need just yes/no answer – whether the verification has detected an error or not. In this case verification can be stopped when the first counterexample is detected in some subspace. The detected counterexample provides sufficient information to claim verification has found an error, and still even a single counterexample points to what exact situation modeling error was detected. As a result, the estimation (46) is precise for the case when all the subspaces are processed, and it is an upper boundary for the case when verification continues until the first detected counterexample.

Also it should be noted that expression (46) deals with numeric context attributes only. Next section provides more general complexity estimations for the case of non-numeric and mixed context attributes.

4.3 Accounting for Non-numeric and Mixed Context Attributes

Section 3.3 points out that non-numeric context attributes can be handled in a following manner: the task should be solved separately for every combination of non-numeric values of all non-numeric context attributes. Consider that in addition to all numeric parameters of formula (46) there is a set of Q non-numeric context attributes, and each of them can take R_q possible values. In that case the task is solved separately for each of $\prod_{q=1}^{Q} \mathbf{R}_q$ combinations of those parameters, and complexity estimation look like formula (47).

$$\text{Complexity: } O(S * N * K * \prod_{q=1}^{Q} R_q) \qquad (47)$$
$$\prod_{i=1}^{N} max_j(L(i,j)+1) \le S \le \prod_{i=1}^{N}(1+\sum_{j=1}^{K} L(i,j))$$

Mixed context attributes can contain both numeric and non-numeric possible values. In the verification approach accounting for mixed context attributes slightly differs from accounting for non-numeric context attribute: the verification task is solved separately for all possible non-numeric values of the context attribute, then it is solved for the case when mixed context attribute is restricted to numeric values. Consider there is a single mixed context attribute, which can take R_{Q+1} non-numeric values, as well as numeric ones. Formula (48) illustrates complexity estimates for that case. When the context attribute takes numeric values, it is counted as context attribute number N+1. Every j-th conjunct have L(N+1,j) breakpoints over that context attribute. The total number of breakpoints over context attribute number N+1, after merging and removing redundant ones, is referred to as L(N+1), just like for any other context attribute.

$$\text{Complexity: } O((R_{Q+1} * S * N * K * \prod_{q=1}^{Q} R_q) + (S' * (N+1) * K * \prod_{q=1}^{Q} R_q)) \qquad (48)$$
$$\prod_{i=1}^{N} max_j(L(i,j)+1) \le S \le \prod_{i=1}^{N}(1+\sum_{j=1}^{K} L(i,j))$$
$$\prod_{i=1}^{N+1} max_j(L(i,j)+1) \le S' \le \prod_{i=1}^{N+1}(1+\sum_{j=1}^{K} L(i,j))$$

In estimation (48) the terms S' refers to the number of subspaces where mixed context attribute is restricted to numeric values. As derived in section 4.1, the number of subspaces $S = \prod_{i=1}^{N}(L(i)+1)$. In the similar manner, $S' = \prod_{i=1}^{N+1}(L(i)+1)$. Therefore, S' = S*(L(N+1)+1) and, consequently, S' ≥ S. As a result, for the first summand of calculation (48) the it is true that $R_{Q+1} * S * N * K * \prod_{q=1}^{Q} R_q \le R_{Q+1} * S' * (N+1) * K * \prod_{q=1}^{Q} R_q$.

Also in (48) for the second summand $S' * (N + 1) * K * \prod_{q=1}^{Q} R_q \leq R_{Q+1} * S' * (N + 1) * K * \prod_{q=1}^{Q} R_q$. Therefore, the order $O((R_{Q+1} * S * N * K * \prod_{q=1}^{Q} R_q) + (S' * (N + 1) * K * \prod_{q=1}^{Q} R_q))$ has the upper bound of $O(2*R_{Q+1} * S' * (N + 1) * K * \prod_{q=1}^{Q} R_q)$, which is $O(R_{Q+1} * S' * (N + 1) * K * \prod_{q=1}^{Q+1} R_q)$. Formula (48) can, therefore, be transformed into formula (49).

$$\text{Complexity: } O(S' * N * K * \prod_{q=1}^{Q+1} R_q) \qquad (49)$$
$$\prod_{i=1}^{N+1} max_j(L(i,j) + 1) \leq S' \leq \prod_{i=1}^{N+1}(1 + \sum_{j=1}^{K} L(i,j))$$

Note that R_{Q+1} in formula (49) is under the multiplication. The interesting note about formula (49) is that mixed context attribute was accounted twice. First it was accounted as a N+1st numeric context attribute, which lead to increased S and N. Second it was accounted as Q+1st non-numeric context attribute, which resulted in additional Q+1st term in the multiplication $\prod_{q=1}^{Q+1} R_q$. The same principle can be applied all involved mixed context attributes. As a result, upper bound of complexity can be estimated as formula (50).

$$\text{Complexity: } O(S * N * K * \prod_{q=1}^{Q} R_q) \qquad (50)$$
$$\prod_{i=1}^{N} max_j(L(i,j) + 1) \leq S \leq \prod_{i=1}^{N}(1 + \sum_{j=1}^{K} L(i,j))$$

Estimate (50) looks much like formula (47), but the notation has different semantics. In formula (50) N is the number of both numeric and mixed context attributes. The same applies to the number of breakpoints L(i,j) – it still means the number of breakpoints of j-th conjunct over i-th context attribute, but now the context attribute can be mixed as well. Also in formula (50) the number Q refers to the total number of non-numeric and mixed context attributes, and the number R_q refers to the number of possible non-numeric values, that q-th context attribute can take. All mixed context attributes are intentionally accounted as both numeric and non-numeric context attributes.

Formula (50) is quite pessimistic upper boundary. However, it is correct and easy to work with. Formula (50), like estimation (46), is also an upper boundary for complexity in case if verification proceeds until the first counterexample. Once a counterexample is found for any combination of non-numeric and mixed context attribute, the verification process can stop.

Formula (50) provides an estimate for a single conjunction. Next section generalizes the estimates for an entire DNF expression under verification and concludes the complexity analysis.

4.4 Complexity Analysis - Summary

In order to complete the complexity analysis, estimation (50) should be generalized to account for multiple disjuncts. The analysis can proceed as follows. Consider that the whole DNF expression under verification contains M disjuncts, each of which is a conjunction of K_i terms, defined over N_i context attributes. According to the developed verification approach, each conjunct can be verified separately, in sequence or in parallel. Formula (50) contains the complexity estimates for every disjunct. The complexity of sequential verification of M disjuncts is presented in formula (51).

$$\text{Complexity: } O\left(\sum_{m=1}^{M} S_m * N_m * K_m \prod_{q=1}^{Q_m} R_{q,m}\right) \quad (51)$$
$$\prod_{i=1}^{N_m} max_j(L_m(i,j)) \leq S_m \leq \prod_{i=1}^{N_m} \sum_{j=1}^{K_m} L_m(i,j)$$

The notation in formula (51) is quite similar to the notation of formula (50). The terms N_m refers to the number of involved numeric and mixed context attributes in m-th disjunct, K_m refers to the number of conjuncts in m-th disjunct and S_m refers to the number of subspaces in m-th disjunct. The same refers to the number of breakpoints: the term $L_m(i,j)$ refers to the number of breakpoints over i-th numeric or mixed context attribute in j-th conjunct of m-th disjunct in the expression under verification. The term Q_m refers to the number of non-numeric and mixed context attributes in m-th disjunct. The value $R_{q,m}$ is the number of non-numeric values, which q-th non-numeric or mixed context attribute of m-th disjunct can take.

It should also be noted that, like formula (50), estimation (51) is an upper boundary of complexity, if the verification proceeds until the first detected counterexample. As lemma 3.2 shows, a counterexample for any disjunct is a counterexample for entire expression, so if a single counterexample is detected in any disjunct, the verification can be stopped.

In order to preliminary estimate the influence of various parameters on the complexity, consider a simplified case where all the disjuncts have similar properties: they have the same number of numeric and mixed context attributes N, the same number of conjuncts K, the same number of subspaces S, and the same number of non-numeric or mixed context attributes Q, each of which can take one of R values. In that case the complexity can be estimated as $O(M*S*N*K*R^Q)$. It points that in average the complexity increases linearly with growing number of disjuncts, although in practice it depends a lot on the configuration of every disjunct: how many conjuncts are there, how many context attributes are involved and how many subspaces of linearity can be generated (which in turn depends on the configuration of breakpoints). The growth of complexity with growing N and K heavily depends on the configuration of breakpoints – in addition to their direct linear influence on formula (51), they can also influence the number of subspaces. For example, adding one new context attribute of interest to one of the situations under consideration will multiply the number of subspaces in that conjunct by the number of breakpoints of newly introduced membership functions plus one. However, adding into account a context attribute, which is already considered by another situation in a conjunct, might have less severe consequences on the total complexity. The dependency on number of non-numeric and mixed context attributes is exponential. The dependency on the geometrical average number of values in mixed or non-numeric context attribute is polynomial.

Despite having some exponential dependencies, the approach works well on the tasks of practical size. One of the reasons is that the breakpoints overlap relatively often due to the features of situation design (see the motivating example). Also complexity requirements are influenced by the fact that verification process should be done only once at the design time. The verification algorithms are prone to numerous enhancements and can be successfully parallelized, but we leave those enhancements out of scope of current article.

To summarize, formula (51) is the final estimation of verification complexity, which takes into accounts all the aspects of verification process. Next section discusses the related work in the area and provides the discussion of the proposed approach.

5 Discussion and Related Work

5.1 Formal Verification of Pervasive Computing Systems

In this article we propose a novel method to verify the context models, based on fuzzy situations. Our previous article [BZ12b] introduced some basic concepts that we use in this article. In the article [BZ12b] we introduced the task of situation models verification, formulated it as emptiness test of an arbitrary situation algebra expression, and solved the emptiness test task for situation models based on context spaces theory. In this article we significantly extend that work and develop verification principles for fuzzy situation models. Fuzzy situations are much more versatile and realistic than original context spaces situations, considered in [BZ12b]. However, the verification of fuzzy situation models is a more complicated task and it requires a completely different emptiness test approach, which has nearly nothing in common with the algorithms defined in [BZ12b]. That approach to emptiness test of situation algebra expressions over fuzzy situations constitutes one of the main contributions of this article.

Pervasive computing research community is showing increased interest in formal verification. However, our article aims to verify the models of situations and their relationships, and the related work aims to verify other aspects of pervasive computing, e.g. agent interaction and behavior rules. As a consequence, this leads to completely different approaches to specification and verification themselves. Usually verification was a method of choice to detect the errors in the behavior specifications of pervasive computing system, while the way pervasive system generalizes incoming context information was out of verification scope.

The paper [IS09] proposed a novel approach to modeling and verification of pervasive computing systems. The authors employed event calculus – an AI formalism [RN06], which allows reasoning about actions and their consequences. Among other uses, in [IS09] the authors used event calculus for expressing undesired behavior of applications, which should be avoided (like "user receives simultaneous audio input from multiple sources"[IS09]). The authors also used assumptions to specify the behavior like "no unauthorized user can enter the room"[IS09]. In the paper [IS09] verification of whether the undesirable situation might ever happen or whether the assumption holds is performed by translating event calculus model into SAT problem, and then applying SAT solvers. The main difference of our article and the approach defined in [IS09] is different aspect under verification. In [IS09] authors aim to verify the behavior of pervasive computing applications, often in terms of activating and deactivating devices and handling user requests. Our approach works on lower level – this article proposes a method to verify whether the transition from sensor values and low level context to further generalizations (i.e. situations) is correct. These generalizations are the basis for behavior of pervasive computing system. In turn, the difference in scope determines completely different approach to verification.

Ambient calculus is designed to specify interaction between multiple agents. The essentials of ambient calculus were introduced in the paper [CG00]. Pervasive systems consist of multiple interacting agents, and some researchers considered ambient calculus as a possible description formalism. The article [CP10] proposes ambient calculus-based approach to specify pervasive healthcare system. The subsequent article [CP11] extended ambient calculus, introduced enhancements to previous approach and introduced verification mechanisms. Once again, the main difference between this article and the approach proposed in [CP10][CP11] is different aspects under verification. Ambient

calculus allows specifying spatial and temporal interactions between multiple agents, which constitute pervasive computing system. Our article proposes a solution to verify the integrity of a context model of a reasoning agent, i.e. whether the agent correctly generalizes context information.

The paper [AC09] considered verification of security, safety and usability properties of pervasive computing systems. For example, possible assertions can be "If a patient is in danger, assistance should arrive in a given time with a probability of 95%"[AC09] or "No component will take an action that it believes will endanger the patient"[AC09]. The authors formalized the properties of pervasive systems in terms of temporal logic. Temporal logic is substantially used in specification and verification of protocols and programs (see [CG99]), and numerous techniques and tools were developed to verify properties expressed in terms of temporal logic. In [AC09] authors also presented an overview of tools, which can be used for verification of temporal logic properties of pervasive computing systems.

The paper [AC09] considered only adaptation and temporal properties of pervasive computing systems. The paper [AC09] did not specify, how the generalizations like "patient is in danger"[AC09] (which is effectively a situation) are achieved or how to verify whether the system detects this situation correctly. In contrast, this article verifies situation models, which are specified and, hence, verified using completely different techniques comparing to [AC09].

The paper [CG09] proposed an approach to verify the action rules of pervasive computing system. The examples of rules, which the paper [CG09] worked with are "If my webcam detects movement then display a pop-up message on my screen and display a message on the screen list"[CG09] or, more complicated example of context-sensitive rule, "If the red button is pressed, then if the webcam recently detected movement inform me with synthesized speech else send me an e-mail"[CG09]. The verification was used, for example, for following purposes.

- Detecting rule redundancy. Redundancy may range from simple repeat to overlap with multiple other rules.
- Making sure that modalities work well together. For example, if the system reports two events simultaneously using speech synthesizers, it can just confuse the user and neither of the reports will be understood.
- Making sure that priorities are handled correctly. High priority messages should be reported first and no lower priority message should take over while high priority message is being reported.

In the paper [CG09] authors proposed a mechanism to derive temporal logic specification from Promela rules, and that specification was used as an input for SPIN [Ho97] model checker.

It should be noted that the approaches proposed in [AC09][CG09][IS09] and the technique presented in this article might complement each other well. They don't overlap in scopes and together they cover verification of both situation awareness and acting on perceived situations.

To summarize, multiple approaches have considered verification as an essential component of pervasive computing development. Different articles proposed verification methods for different aspects of pervasive computing systems. Different aspects under verification resulted in different required approaches to verification and specification.

5.2 Fuzzy Logic for Context Awareness in Pervasive Computing

The concept of fuzzy situations, used in this article, was inspired by fuzzy situation inference (FSI) concept proposed by Delir et. al. [DZ08]. In [DZ08] the authors proposed a combination of context spaces theory approach to context modeling and fuzzy logic-based approach to situation awareness. The paper [DZ08] also introduced an extension to account for sensor uncertainty. In [DZ08] the authors applied newly proposed situation awareness mechanisms for health reasoning on ECG-capable mobile device. In this article we extend the situation model, proposed in [DZ08], in order to account for non-numeric sensor readings and possible sensor unavailability. Fuzzy situation concept proposed in [DZ08] is a subset of fuzzy situation models used in this article, so the proposed verification techniques can without any modification be applied to non-extended FSI models.

The paper [DZ08], in turn, extends the context spaces approach, described in [PL08a]. Original context spaces theory also features situation algebra based on Zadeh operators, and the situations in original context spaces approach are modeled as weighted sum of contributions. However, original context spaces approach uses piecewise constant as the contribution function (contribution function is replaced by fuzzy membership function in [DZ08] and this article). The verification methods provided in this article are not suitable for original context spaces situation models. However, the suitable verification mechanisms were proposed and proved in our previous works [BZ11c][BZ12b].

Multiple related work examples applied fuzzy control process for context awareness and acting on context information. For example, the paper [MS02] applied fuzzy logic to context aware control of mobile terminals. In the paper [CX05] the authors applied fuzzy control to the adaptation of pervasive services. In the paper [AG10] authors used fuzzy engine to control the actuation of proper services. Fuzzy situations in this article were inspired by fuzzy logic techniques, and many aspects of fuzzy context awareness can be represented in terms of fuzzy situations. For example, membership functions (figure 2) are widely used membership functions for fuzzy sets [HM93], and therefore membership in fuzzy set is effectively the same as fuzzy situation over a single context attribute. Therefore, multiple aspects of fuzzy context awareness can be translated into fuzzy situation terms and then verified by the approach proposed in this article.

To summarize, the context model, presented in this article, in some aspects resemble various classes of context awareness mechanisms used in related works. Therefore, verification technique, presented in this article, is likely to be applicable to multiple classes of existing context awareness methods with minor adaptation efforts. The exact methods of adaptation and required modifications are the subject of future work.

6 Conclusion and Future Work

In this article we propose and prove a novel method for formally verifying correctness of fuzzy situation models. We extend and enhance the notion of fuzzy situations in order to achieve robust situation awareness and mitigate the consequences of unavailable sensor data. In this article we also prove that verification by assertion of emptiness is applicable for fuzzy situations. We propose and prove a novel step-by-step approach for emptiness test of a situation algebra expression, which is the core of our verification technique. As part of this work we implement the verification approach as an extension for ECSTRA [BZ11a] context awareness framework, and analyze practical efficiency of the proposed method. We analyze complexity of the verification approach and discuss its applicability to other context awareness approaches. The proposed methods can impact the design and running of

pervasive computing systems in a major way due to their potential of detecting pervasive system design mistakes, which are not detected by testing, and due to their capabilities of proving that certain kind of error is not present in system design. Two major future work directions include: improving the fuzzy situation verification method; improving general aspects of situation models verification. Future improvements of the proposed approach include the following research directions.

Efficient ordering of subspaces and combinations. The subspaces of linearity and non-numeric combinations, which are more likely to contain counterexamples, should be generated and analyzed first. It will significantly reduce the running time for the use cases when verification proceeds until the first counterexample is computed.

Parallelization. Efficient parallelization can significantly improve the performance of verification algorithms, since different subtasks of the verification task can be run in parallel.

Verification in general can be improved in following ways.

Verification of temporal properties. Current verification approach cannot analyse timing dependencies between situations. Improvement of verification techniques to incorporate temporal properties can be of much use to pervasive computing area.

Automated repair. If the verification has detected an error, even with the counterexamples it can be complicated to narrow it down and repair the definition mistake. It can be highly beneficial if the repair is suggested by the verification algorithm itself. The suggestion for the repair should take into account all the properties that should be maintained, not only the one that caused an error.

Automated situation generation. This future work direction is the continuation of previous task. As an input, the developer specifies multiple requirements that situations should comply with. Those can be emptiness assertions like described in section 2.3, or expected testing results, like "if the light level is above 500Lx and noise is below 30dB, situation *ConditionsAcceptable(X)* should have certainty1". The expected output is a list of situation specifications. For example, for fuzzy situations it can be a list of involved context attributes, weights and membership functions.

Chapter VIII

Context Prediction in Pervasive Computing Systems: Achievements and Challenges

Based on:

1. Boytsov, A. and Zaslavsky, A. Context prediction in pervasive computing systems: achievements and challenges. in Burstein, F., Brézillon, P. and Zaslavsky, A. eds. *Supporting real time decision-making: the role of context in decision support on the move*. Springer p. 35-64. 30 p. (Annals of Information Systems; 13), 2010.[9]

[9] This chapter is based on the article [BZ10a]. The fragments of the arctile [BZ10a] that were included in chapter I are omitted from chapter VIII.

Foreword

Previous chapters addressed the problems of defining situations and checking that the definition is correct. Therefore, previous chapters answered the research questions 1 and 2. Chapters I-VII also built a solid foundation for investigating subsequent research question: how to predict future situations and how to properly act according to prediction results? This chapter answers the first part of the research question 3: how to predict future situations? Chapter VIII represents a survey of context prediction research area and introduces the classification of context prediction approaches. This chapter also allows concluding that context prediction efforts are mainly focused in situation prediction area. Also chapter VIII identifies open challenges of situation prediction, and those challenges will be addressed in subsequent chapters.

Context Prediction in Pervasive Computing Systems: Achievements and Challenges

Abstract. Context awareness is one of the core principles of pervasive computing. Context prediction is also recognized as a challenge and an opportunity. Numerous approaches have been applied to resolve context prediction. This work develops and justifies the principles to analyze and compare context prediction methods, analyses the development in the area, compares different context prediction techniques to identify their benefits and shortcomings, and finally identifies current challenges in the area and proposes the solutions.

Keywords: Pervasive Computing; Context Awareness; Context Prediction; Sequence Prediction; Markov Model; Bayesian Network; Neural Network; Branch Prediction; Expert System.

1 Context and Context Prediction

Pervasive computing paradigm is a relatively recent approach where computing systems are integrated into everyday life and the environment in a transparent, graceful and non-intrusive manner. An example of a pervasive computing system can be a smart home that adjusts lights and temperature in advance before the user enters the room, which increases the efficiency of energy and water consumption. Or it can be an elderly care system that decides whether the user needs advice or assistance. Or it can be a smart car that proposes the best route to the destination point and that assesses its own condition and provides a maintenance plan. Many pervasive computing systems are now being introduced into our lives.

One of the grounding principles of the pervasive computing system is context awareness. Earlier works on context and context awareness proposed numerous different definitions of the context. A. Dey and G. Abowd [DA00], performed a comprehensive overview of those efforts. From now on their definitions of context and context awareness will be used: "Context is any information that can be used to characterize the situation of an entity" [DA00]. So, in fact, every piece of information a system has is a part of that system's context.

"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task" [DA00]. Or, in more simple words, the system is context-aware if it can use context to its benefit. Context awareness is one of the main principles of pervasive computing.

Reasoning about the context is the process of obtaining new information from the context. Context model, in that case, becomes an intuitively understandable term that can be formally defined as a way of context representation that is used for further reasoning. Context prediction, therefore, does not need any special definition and merely means the activities to predict future context information.

Context awareness and context prediction are relatively new areas. However, they already have some methods developed for reasoning about the context and prediction of the context.

Cook and Das [CD05] give a quite comprehensive overview of the context prediction area with a focus on smart home environments and location prediction. Pichler et al. [PB04] provide a context prediction overview, focusing on artificial intelligence-based techniques; they discuss not only the examples, but further possibilities as well. [Z08] contains another context prediction overview with the focus on user behavior modeling; Hong et al., [HS09] focuses on user preferences inference and utilizing the context history; Cook et al., [CA09] focuses on ambient intelligence systems; and Boytsov et. al [BZ09] focuses on machine learning based techniques. One of the most comprehensive works regarding general approach to context prediction is [Ma04a], which will be mentioned further in more details.

2 Context Prediction in Pervasive Computing

2.1 Context Prediction Task

Context awareness is one of the core features of any pervasive computing system. Context prediction is acknowledged as both a challenge and an opportunity. Some works provide comprehensive lists of context prediction use case scenarios. Those use cases include (list partially based on [NM05][Ma04a]):

1. Reconfiguration. Sometimes configuration-related tasks take a while to complete. This includes installation of updates, loading and unloading libraries, starting new applications, processing the infrastructure changes related to node mobility, searching in large databases. If the system can predict the need for those tasks in advance, it can perform the work beforehand and avoid unnecessary delays.

   An application specific example was presented by Mayrhofer, [Ma04a]. When the key appears near upper-class BMW cars, the car predicts that it is going to be started and the on-board navigation system initiates boot-up. Therefore, when the user enters the car, the navigation system is already fully functional. Otherwise, it would have taken extra 30 seconds to complete.
2. Device power management. Device that are unused and will not be used in near future can be shut down or switched to sleep mode.

   There are other scenarios that fall under that category as well. For example [NM05], if the user attempts to send a large multimedia message while in an area of bad radio reception, the system can predict that the user is going to enter a better reception area soon and will delay sending the message (and therefore saving power).
3. Early warning of possible problems. Context prediction can determine whether the system is about to enter an unwanted state and act accordingly. For example, a pervasive system can predict that it is going to run out of memory or computation power soon and act proactively to counter that problem – for example, find the devices to share the computations, offload the data and drop unnecessary applications. Or, for example, a pervasive system can predict that user is going to enter a traffic jam soon. In that case, the system can find a way around the traffic jam and provide it to the user before the traffic jam area is entered. Different cases of accident prevention also fall under the category of early warning of possible problems.
6. Planning aid. If a user's needs can be predicted, a pervasive system can meet them proactively. For example, an air conditioner in a smart home can be launched in advance to have a certain air temperature when the user returns from work. Or in a smart office, the door can be opened right before the person enters.

7. Inferences on other's future contexts. User can actually be influenced by the future context of another user. For example [NM05], a user may have confidential information on the screen. Therefore, when someone passes by, the screen should be hidden. Prediction of other people's interference can help to hide the image proactively and in a timely fashion.
8. Early coordination of individuals. This can be viewed as a consequence of the previous point. If the needs of several users in a group can be predicted, the system can act to satisfy the interests of the group as a whole.

Future context can be predicted using a large variety of models. However, implementation of context prediction faces challenges that distinguish context prediction from many other kinds of prediction tasks. The challenges include the following (partially based on the papers [NM05] [Ma04a]):

1. Pervasive systems work in real time.
2. Pervasive systems need to predict human actions. This is one of the main reasons why most of context prediction techniques are grounded in machine learning. Human actions depend on human habits and personal features which, in turn, often cannot be guessed beforehand but can be inferred during the run time.
3. The systems work in discrete time. All context data are provided by sensors and sensors work in an impulse manner which provides the measurements in certain points of time.
4. The data are highly heterogeneous. Lots of data of different nature and different type are coming from different sources. For example, context data can be numerical and non-numerical; context data can come periodically or when a certain event occurs.
5. Sometimes hardware capabilities are limited. Using lots of small and preferably inexpensive devices is very common in pervasive computing. Many devices have to be relatively autonomous and use wireless interfaces for interactions. Devices of that kind have limited power supply and limited computational capacity.
6. Connectivity problems are possible which can cause problems including data loss and sensor unavailability.
7. The learning phase should be kept to a minimum. Often a pervasive computing system needs to start working right away. Ability to incorporate prior assumptions about the system is also highly desired in order to achieve a fast system start.
8. Sensors are uncertain. Not taking that into account can lead to reasoning and prediction problems.
9. There is a need for automated decision making.

So far there are some works that address context prediction challenges, but there still is a definite lack of universal approaches to the problem.

In this chapter context prediction techniques will be classified according to the formal models and approaches which inspired them. This is an insightful way that can provide a direction for new techniques research. Sometimes those approaches can overlap and, therefore, some approaches can be associated with several classes.

2.2 From Task Definition to Evaluation Criteria

To develop a basis for context prediction methods classification, we need a formal definition of the context prediction task. It will help to identify criteria for classification and evaluation of context prediction approaches.

Let S_1, S_2, be the context data at a certain moment in time. As usual, every portion of context data S_i is a vector that contains sensor data and the results of sensor data preprocessing.

A context predictor in most general sense can be viewed in following manner (formula (1)).

$$Pr = G(S_1, S_2, \ldots, S_t) \quad (1)$$

In formula (1) t is current time and Pr is prediction result, the prediction result can be either just a set of expected future context features, or their distribution, or a future event, or a future state. Context prediction operation is represented by operator G. Initial knowledge and assumptions about the system are also included in G. To summarize, context prediction results depend on the context prediction method and possibly on context history and current context data. Questions of context history awareness will be addressed later in this section.

In more details, the context predictor can be viewed in following manner (see Figure 1).

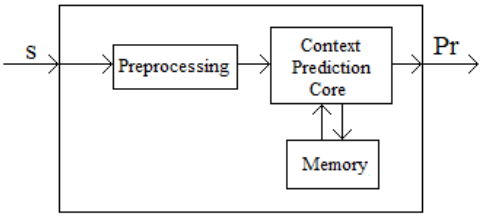


Fig. 1. Context prediction – general structure

Several parts of the context predictor can be identified.

Part 1. Context prediction core. It implements the exact context prediction method. This can be an ad hoc method defined for an exact case or any kind of well-known approach (like neural network, Markov model, Bayesian network, sequence predictor). Actually, the method used as the context prediction core is the main criteria of distinguishing one method of context prediction from another.

To define the context prediction method, we need to define its main principle (e.g., Bayesian network, Markov model, neural network) and its parameters. Those parameters can be, for example, transition probabilities for Markov chain, neural network coefficients, or distributions for Bayesian network. The context predictor can obtain the parameters as prior knowledge or infer the parameters during run time. Here are the suggested evaluation criteria:

Criterion 1. Determine whether prior estimations about a pervasive system can be incorporated into the method. If the method cannot do that, it might result in low effectiveness at the startup. Sometimes prior training can be a workaround for that problem: to pre-train the context predictor, the trainer system needs to generate training data

according to prior estimations and present it as pre-training before the system actually starts.

Criterion 2. Determine whether prior estimations about the pervasive system can be incorporated into the method. In pervasive computing systems the problem of having numerous unknown parameters is quite common. For example, pervasive computing systems often involve user behaviour prediction. User behaviour depends on user habits; the system usually cannot guess those habits in advance and needs to learn them during the run time. Practically, in pervasive computing there are almost no methods that are incapable of incorporating run time knowledge.

Criterion 3. Determine whether white box/black box. Another criterion, closely correlated with two aforementioned ones, is whether the method is a "black box" or "white box." If the method is a white box, method parameters usually show in a quite insightful manner how the system really works. By looking at system parameters the expert can tell what exactly the system has learnt or, in turn, having some prior estimation, the expert can configure the system parameters accordingly. If the method is a black box, it is capable of prediction, but an expert cannot see the underlying reasons for the current prediction even if that expert knows the complete set of method parameters. Black box methods are generally unlikely to be able to incorporate prior knowledge about the system.

For example, transition probabilities of the discrete time Markov chain explicitly reveal the chance for the system context to be in a particular state at a particular time; the state, in turn, corresponds to certain context features. And, having the transition probability, expert can easily understand what kind of regularity is found (at least in terms like "if context at current time t has features a_1, a_2, a_3, …, that means that it will have features b_1, b_2, b_3, … at time $t+t'$ with probability p"). So, Markov chain-based methods are white box methods. As for neural networks, even though the expert knows all the weights for every neuron, it is usually impossible to tell what kind of regularities it corresponds to. So, neural network-based methods are black box methods.

Criterion 4. Determine whether estimation of prediction reliability is incorporated into the method. In practice if the method is capable of estimating its own reliability, the predictor usually returns the distribution of predicted value (e.g., Markov models, Bayesian networks), not just the value itself. If reliability estimation is not possible in the method, such method usually returns just predicted value with no probabilistic estimations.

Criterion 5. Determine outlier sensitivity. All sensors have some degree of uncertainty. Moreover, the sensors can become unavailable or the measurement results can be lost in transfer. In that case, an outlier appears. The outlier can significantly alter the prediction results. In a very general sense we can classify outlier sensitivity in the following groups.

- *No sensitivity*. Outliers will not affect prediction effectiveness in any way. No examples so far. Theoretically it is possible if, for example, we have just a prediction formula which does not take any current data into account (formula (2)).

$$Pr = G(S_1, S_2, \ldots, S_t) = G(t) \quad (2)$$

But practically that kind of predictor is very inflexible and, therefore, is not used.

- *Moderate sensitivity*. Outliers do affect prediction effectiveness, but over time the influence of the outlier fades. For example, the neural network of the Markov predictor learns over time, and if there is an outlier in the training sequence, it will

have an effect. However, the amounts of data in the training sequence will be growing and the influence of the outlier will be decreasing.

- *High sensitivity*. One outlier can significantly influence subsequent results of context predictions; the effect of the outlier will not decrease until the outlier value is completely excluded from the history. For example, the values of sensor measurements over time can be treated as function and interpolated. Later that function can be extrapolated to the future and it will be the prediction result. But outlier presence can significantly alter the resulting function and the effect will not be reduced until that point is out of the history range (which will happen practically, at least due to memory limitations).

Criterion 6. Determine what types of incoming data are supported by the context predictor. Context can contain data of several types: real, integer and non-numerical (e.g., room in smart office or the position of the switch). If some data type of the context is not accepted by the context prediction method, preprocessing needs to be performed.

Part 2. Preprocessing block. The preprocessing block transfers incoming sensor data to the format that is applicable to the context prediction core. In a very general sense it can be represented in the following manner:

$S' = Prep(S)$

Where S – current received context data, S' – preprocessed context data for further usage by context prediction core. Prep() represents preprocessing operator. Preprocessing operation also can theoretically be dependent on historical data, but practically it is not likely.

Criterion 7. Determine preprocessing information loss. The more information lost during preprocessing, the greater the chance to miss significant information, not use it during context prediction and therefore get reduced context prediction capability. So here is one more context prediction method evaluation criterion: whether the information is lost during preprocessing.

- No information loss. For example, context can be left as is (it can be represented as state-space model with later prediction using filter theory or context data can be used directly as an input for the neural network). Or new context attributes can be introduced during preprocessing (e.g., one of the environmental characteristics can be estimated in two different manners to detect sensor failure or use filtering to combine two sources of data). So, processing in this case can require some efforts as well. The criterion for the method to be included in this category is following: for every S' there should be at most one value of S.
- Information loss present. Denotes all other cases. For example, some values can be aggregated, or context can be completely reduced to a finite number of states of the Markov model or probabilities of Bayesian network nodes, or the context can be decomposed into event flow and timing information can be lost, and many more examples.

Usually the presence of information loss depends both on the prediction method and on sensor data the system obtains. For example, if there are only a few sensors with a small set of discrete non-numerical data coming from them, the Markov model can be created without information loss – every predictor state (S') will correspond to every possible combination of sensor values (S). However, even if the system has just one real-valued sensor, creating the Markov model without information loss is impossible – there is an

infinite number of possible values of S and it cannot be covered by any finite number of states in S'.

Part 3. Memory. The predictor might need to store history or parameters in an applicable manner. Some methods require only current value and do not store history in any manner. Some of the methods are capable of handling all the history and using it to its benefit.

Criterion 8. Determine constant or variable amount of memory needed. For example, a neural network needs only memory to store weight coefficients for neurons. The Markov model needs only a fixed amount of memory to store transition probabilities and some intermediate data to obtain them. However, an expert system-based context predictor that constantly introduces new rules or a sequence predictor-based approach with growing prediction tree requires a variable amount of memory and the memory demand can grow over time.

In summary then, here are the defined criteria for evaluation of context prediction methods:

1. Prior knowledge accumulation? Yes / No.
2. Real-time knowledge accumulation? Yes / No.
3. "Black box" / "White box"?
4. Prediction reliability estimation? Yes / No.
5. Outlier sensitivity? No / Medium / High.
6. Types of data supported?
   - 6.1 Real? Yes / No.
   - 6.2 Integer? Yes / No.
   - 6.3 Non-numerical? Yes / No.
7. Information loss on preprocessing? Yes / No (usually it depends on certain conditions).
8. Memory amount needed? Fixed, Variable.

3 Context Prediction Methods

In practice, the methods used most frequently are:

1. Sequence prediction approach.
   This approach to context prediction is based on the sequence prediction task from theoretical computer science and can be applied if the context can be decomposed into some kind of event flow.
2. Markov chains approach.
   Context prediction techniques based on Markov chains are quite widespread. Markov chains provide an easily understandable view of the system and can be applied if the context can be decomposed into a finite set of non-overlapping states.
3. Bayesian network approach.
   This can be viewed as the generalisation of the Markov models. It provides more flexibility but requires more training data in turn.
4. Neural networks approach.
   Neural networks are biologically inspired formal models that imitate the activity of an interconnected set of neurons. Neural networks are quite popular in machine learning. Context prediction approaches based on neural networks exist as well.
5. Branch prediction approach.

This approach initially comes from the task of predicting the instruction flow in a microprocessor after the branching command. Some context prediction systems use similar algorithms.

6. Trajectory prolongation approach.
   Some context prediction approaches treat the vector of context data as a point in multidimensional space. Then the context predictor approximates or interpolates those points with some function, and that function is extrapolated to predict future values.
7. Expert systems approach.
   Based on expert systems and rule-based engines, the expert systems approach appears in some works on context prediction. The goal of the approach is to construct the rules for prediction. It provides a very clear view of the system.

Subsequent chapters address those approaches in more details.

3.1 Sequence Prediction Approach

The sequence prediction task is a quite researched problem in theoretical computer science. Generally, the problem is as follows: having the sequence of symbols <S(1), S(2), …, S(t)>, received at the time from 1 to t, the task is to predict next symbol S(t+1).

If the context of pervasive computing systems can be represented as a flow of symbols (e.g., flow of events or state of Markov model in particular time), the context prediction problem can be viewed as a sequence prediction task.

D. Cook and S. Das [CD05] provided quite a comprehensive overview of sequence prediction techniques used in context prediction (particularly in smart home environments).

So far the earliest works which treated the problem of user activity prediction as sequence prediction tasks were the works of Davidson and Hirsh, related to command prediction in the UNIX environment [DH97, DH98]. Those works inspired subsequent research in the area of user activity prediction, including context prediction in pervasive computing systems and particularly smart home systems.

In a number of works [Ma04a][BD09][DC03][RB03] algorithms from the Lempel-Ziv family were suggested as a means of context prediction. In practice, the LZ78 algorithm was a baseline for a set of context prediction algorithms. For more details refer to Ziv and Lempel [ZL78] for the description of the algorithm; to Feder et al. [FM92] for subsequent researches; and to Gopalratnam and Cook [GC03] for the overview of enhancements related to context prediction.

Bhattacharya and Das [BD99] introduced the LeZi update, which addressed the problem of predicting the next location of the user in a cellular network. Proactivity in that case could benefit the cellular system by enhancing paging and location updates. The paper [BD99] also inspired the subsequent work of Das et. al. [DC02], where the authors used the LeZi update algorithm for the smart home environment. The work was further enhanced in Roy et al. [RB03], where the authors addressed the questions of energy consumption prediction based on path prediction.

Compared to the classical LZ78 approach, the LeZi-Update approach tried to keep a track on all contexts within a phrase. The idea of enhancement is as follows: on every step update frequency – not only for every prefix like in LZ78, but for every suffix of this prefix. E.g., if the received phrase is "AB**C**", then the frequency should be updated not only for "ABC", but for "C" and "BC" as well.

LeZi-Update inspired further enhancements including Active LeZi. Gopalratnam and Cook [GC03] introduced the Active LeZi algorithm in order to predict the actions of

inhabitants in smart home environments. Authors identified several shortcomings of the initial LZ78 algorithm:

1. The algorithm will lose any information that will cross the boundary of the phrase.
2. There is a low rate of convergence.

The authors proposed a solution to the shortcomings. Using the LeZi Update as a basis, they introduced a sliding window of previously seen symbols to be able to detect patterns that cross phrase boundaries. During a testing scenario without any noise or sensor uncertainty on highly repetitive data, Active LeZi reached almost 100 per cent prediction rate after around a 750-symbol training sequence. However, when noise was introduced into testing scenarios in practical implementations, the prediction rate began to float around 86 per cent. Testing on real data has shown that Active LeZi has around 47 per cent prediction rate after around 750 symbols. Mayrhofer [Ma04a] suggested using Active LeZi in a more general case of context prediction task. In Section 4 more details on that work will be given.

Therefore, sequence prediction turns out to be a feasible and widely used approach for context prediction and this approach has shown good performance practically. Moreover, some theoretical enhancements in the sequence prediction area were made while researching the context prediction problem (e.g., Active LeZi development). However, the sequence prediction approach has several shortcomings. Generally, reduction of the entire context to the mere sequence of symbols introduces the possibility of losing valuable information on preprocessing. Another problem is that the system cannot deal with levels of confidence when a situation has occurred. Prediction reliability can be computed, but the reliability of observed data cannot be taken into account. One more major shortcoming is that this approach generally does not deal with time (although some exceptions do exist). It won't detect timing-dependent regularities like "If engine is overheated, it will break down in 10 minutes". The duration of a situation occurring can also be important. It is not considered in the sequence learning approach as well. However, although the sequence prediction approach has its shortcomings, it still turns out to be a good choice for many practical tasks.

3.2 Markov Chains for Context Prediction

The discrete time first order Markov chain is a model that consists of following parts:

1. Finite number of possible model states: $S = \{S_0, S_1, \ldots, S_{n-1}\}$.

2. The probability of the system to transition from one state to another is presented in the expression (3).

$$P_{ij} = P(S(t+1) = S_j \mid S(t) = S_i),\ i,j \in [0..n-1],\ t=0,1,2,\ldots \quad (3)$$

Markov property is the property of the system's future states to be dependent on the current state only (and not on the history). In a more general case of Markov chains of order K, transition probabilities depend not only on current state, but on the history of states down to time $t-(K-1)$. From now on we are going to refer only to discrete time Markov chains unless explicitly stated otherwise.

The transition probabilities are stationary – P_{ij} does not depend on time.

3. Initial probability of the model to be in certain state: $P(S(0)=S_i)\ i = 0,1,\ldots,n-1$

Pervasive systems usually employ discrete time models due to the nature of sensory originated data, which usually arrive at certain moments in time, either in some period or at the occurrence of some event. So from now on we are going to refer only to discrete time Markov chains, unless explicitly stated otherwise.

Markov chains are widely researched formal models that were applied to numerous practical tasks. See, for example, Russel and Norvig [RN06] for more details on Markov chains, and Baum and Petrie [BP66], Rabiner [R90] on more details on hidden Markov models.

Some projects applied Markov chains to address context prediction problems. For example, in Kaowthumrong et al. [KL02] addressed an active device resolution problem. A device resolution problem is a problem when the user has a remote control to interact with a set of devices, but there are always several devices in user proximity. The system needs to determine to what device the user is referring. The proposed solution was to predict the next device to which the user is likely to refer and use this information to resolve ambiguity on the next step. The authors proposed the hypothesis that the next device depended on the current device only. That hypothesis is actually Markov property. The context prediction system is built into the Markov chain; devices show states and transition between the states representing the order of device access. E.g., if the user turns on the light (state L) and then turns on the TV (state T) that means that the model transitioned from state L to state T. The model continued being in that state until a new action was performed by the user. The goal was to infer user habits from observed action sequence. Initially, transition probabilities were unknown. During the run time, the system inferred transition probabilities by calculating relative transition frequency (expression (4)).

$$P_{ij} = \frac{N(\forall t>0, S(t-1)= S_i, S(t)=S_j)}{N(\forall t\geq 0, S(t)= i)} \quad (4)$$

In the formula (4) is the count of cases.

Having several devices in proximity of remote control, the system chose the one with the highest probability among them. Prediction accuracy was estimated at 70-80%.

Another example is the work by Krumm [Kr07], where the author used discrete time Markov chain for driving route prediction. Road segments were states in Markov chain and the transitions were the possibilities to enter another road segment e.g. at the crossroad. Probabilities were inferred from the history in a manner similar to Kaowthumrong et. al., [KL02]. Author used Markov chains of different orders and compared the results. One-step prediction accuracy exceeded 60% for 1st order Markov model. Prediction accuracy exceeded 80% and tended to 90% when Markov model order grew to 10. Overall, the system was able to predict one segment ahead (237.5 meters in average) with 90% accuracy and three segments ahead (712.5 meters in average) with 76% accuracy. Zukerman et. al. [ZA99] used Markov models for prediction of next user request on WWW.

Using discrete time Markov chains in context prediction is plausible and easy way when the system is fully observable and when the context can naturally be represented as a limited set of possible states changing over discrete time. However, more complicated cases require the extension of Markov chain approach.

Hidden Markov model extends Markov chains to the case of partial observability. Hidden Markov models (HMM) were introduced in the work [BP66]. HMM can be defined as Markov models, where the exact state of the system is unknown. One of very popular example is urns with colored balls (according to [R90], introduced by J. Fergusson in his

lectures on information theory). Each urn contains many balls of different colors, and the distribution of colors is different between the urns. The user sees the color of the ball, but she does not know which urn the ball is taken from. In this example exact urn stands for hidden state and the color of the ball stands for output. Comparing to fully observable Markov chain description, HMM also requires following distribution (expression (5)).

$$P'_{jk} = P\,(Y(t) = k \mid S(t) = S_j) \tag{5}$$

In expression (5) the term P_{jk} represents the probability of seeing output k in state j at time t.

Several tasks are quite common for HMM:

1. Given the model and the output, identify underlying sequence of states and transitions.
2. Given the model and the output, identify the probability of an output to correspond that model. One of the special cases of that task is following: given a set of possible underlying models, choose a model that matches the output sequence best.
3. Given a model without transition probabilities and given some output, find the missing probabilities. The process of detecting those probabilities is usually referred as training HMM. This HMM use case is the most common one for context prediction task.

For more information on the algorithms for different tasks, refer, for example, to the work by Rabiner [R90].

HMMs were used for context prediction in several projects. For example, Gellert and Vintan [GV06] used hidden Markov models to obtain the prediction of the next room a person was likely to visit. The resulting HMM represented every room as a possible state. Simmons et al. [SB06] the authors used HMMs for route prediction. The prediction system represented the road structure as a graph where nodes were crossroads and edges were roads between crossroads. The state of HMM was the combination of the position on the road (which was observed) and destination point (which was not observed, but could be guessed). According to the authors, prediction accuracy was up to 98 per cent in most cases. Hidden Markov models are rather popular for the cases when the Markov model is applicable and the system is partially observable; partial observability often appears either due to sensor uncertainty or due to taking into account such parameters as user intentions or user emotions which are not directly measureable.

One more extension of the Markov chain is the Markov decision process (MDP). MDP is a formal model that consists of following parts:

1. Finite number of possible model states: $S = \{S_0, S_1, \ldots, S_{n-1}\}$.
2. A set of possible actions to be taken in state S_i: $A(i) = \{a_{i1}, a_{i2}, \ldots\}$, $i = 0,1,\ldots,n-1$.
3. The probability of the system to transition from one state to another on particular action is presented in expression (6).

$$P_{ij}(k) = P(S(t+1) = S_j \mid S(t) = S_i\,,\, a(t) = k),\ i,j \in [0..n-1]\ ,\ t=0,1,2,\ldots \tag{6}$$

4. Initial probability of the model to be in certain state: $P(S(0)=S_i)$ $i = 0,1,\ldots,n-1$.
5. A reward or cost for transferring from state S_i to S_j on choosing certain action a: $R(S_i, S_j, a)$.

There are often considered simpler cases when the reward depends only on next state and action ($R(S_j, a)$) or even next state only ($R(S_j)$).

Partially observable MDP (POMDP) is an extension of MDP idea for partially observable systems. POMDP extends the concept of MDP in the same manner like HMM extends the concept of the Markov chain. POMDP also requires the distribution of possible observations depending on the state (formula (7)).

$$P'_{jk} = P\,(Y(t) = k \mid S(t) = S_j) \tag{7}$$

Usually the goal of MDP processing is to find a policy – principles of choosing the action to maximise the rewards. Policy can be defined as $\pi = \{\pi_0, \pi_1, \pi_2, \ldots\}$, where $\pi_i = \pi_i(h_i)$ is a probability distribution of choosing certain actions depending on observed history.

Sometimes the objective is to resolve an inverse problem with one or both of the following considerations: the system needs to find cost or reward function R(i,a) that explains user behaviour. This task is usually referred to as apprenticeship learning. Refer to Abeel and Ng, 2004 for more details on the apprenticeship learning problem.

To compare different policies several methods are available:

1. Discounted expected total reward (expression (8)).

$$R = R(S_0) + \sum_{i=1}^{L} \beta^i R(S(i-1), S(i), A(i)) \tag{8}$$

In formula (8) the term $\beta \in [0,1]$ is a discount factor, S(t) stands for the state at time t, A(t) stands for action at time t, R stands for reward function and L stands for horizon (it is often infinite).

1. Average reward criterion (expression (9)).

$$R = \lim_{N\to\infty}\left(\frac{R(S_0) + \sum_{i=1}^{N} \beta^i R(S(i-1), S(i), A(i))}{N}\right) \tag{9}$$

Refer to the book by Russell and Norvig [RN06] for more details on MDP theory.

Markov decision processes were used in location prediction and gained some popularity in driving route prediction. For example, Ziebart et. al., [ZM08] proposed PROCAB method (Probabilistically Reasoning from Observed Context-Aware Behavior) and used it for vehicle navigation. More specifically, authors addressed three issues: turn prediction, route prediction and destination prediction. The system represented user behavior as sequential actions in Markov decision process. Authors adopted apprenticeship learning approach: at first, system observes driver actions and infers driver preferences. After some training the system is able to predict driver actions. Inferred cost function is used to compare the significance of route benefits and shortcomings according to the opinion of the driver (which is very different from person to person). For apprenticeship learning, authors adopted the approach proposed by Abbeel and Ng [AN04]. As a result, system became capable of predicting route, turn or destination and providing services and suggestions proactively. Hoey et. al. [HP07] authors used partially observable Markov decision processes in elderly care systems for people with dementia. The system tracked the process of handwashing of people with dementia and decided whether to call a caregiver, give and advice or just do not interfere. Using the camera, elderly care system could observe some information like the stage of handwashing process (e.g. turned the water on, watered the hands, soaped the hands) , and some information like the stage of the disease was considered to be hidden state.

So, Markov decision processes are plausible and practically effective way to predict the context in the situations when Markov models are applicable and control actions can significantly affect prediction results.

3.3 Neural Networks for Context Prediction

Neural networks are formal mathematical models that imitate biological neural structures. Starting back in the 1940s with the first models of neuron, it became one of the most popular ways of solving artificial intelligence related tasks. Learning capability allows neural networks to solve a variety of problems including pattern association, pattern recognition, function approximation. For comprehensive neural networks overview refer, for example, to the works by Russell and Norvig [RN06][RN09] or to the book [Ha09].

Neural networks were used for context prediction as well. For example, Mozer [Mo98] described smart house that predicts expected occupancy patterns in the house, estimates hot water usage, estimates the likelihood for a zone to be entered soon, etc. In that project authors used feedforward neural networks trained with back propagation. Indoor movement prediction related projects considered neural network approach as well. For example, in the work Vintan et. al. [VG04] addressed the problem of finding next room the person is going to visit. Predictor took current room number as an input and was expected to give most probable next room number as an output. For prediction method authors chose multi-layer perceptron with one hidden layer and back propagation learning algorithm. System used log of movement for training. Al-Masri and Mahmoud [AM06] authors suggest to use artificial neural networks for providing mobile services. Authors presented SmartCon application that is capable of learning user needs and dynamically providing applicable mobile services. Authors elicited relevant information by training neural network with device-specific features (all the information about user's device: hardware type, terminal browser, software platform), user-specific features (learned user preferences) and service specific-features (service provider preferences). Later this information is user to suggest proper mobile service to user. Lin et. al. [LW08] suggested to use neural network for smart handoff. The process of handoff occurs when mobile device is moving away from coverage area of one base station under coverage area of another. Handoff decision is usually based on such characteristics like signal strength, bit error rate, signal to noise ratio. However, sometimes user is moving close to the borders of coverage area of different base stations and lots of unnecessary handoffs appear. The goal is to predict whether user is likely to move under the coverage area of any base station completely. Prediction results will affect handoff decision. Authors proposed to use multilayer perceptron to detect the correlation between packet success rate and a certain set of metrics. According to Lin et. al., [LW08] their algorithm outperforms current common handoff algorithms.

As a result, neural networks turned out to be a feasible way of context prediction for many practical use cases. There are various types of neural networks available and they can provide a flexible tradeoff between complexity and effectiveness. The major shortcoming of the neural network approach is that it is a black box – by examining neural network structure it is not possible to say what exact regularities are detected.

3.4 Bayesian Networks for Context Prediction

The approach of Bayesian networks generalizes Markov models and avoids some of the Markov model shortcomings. For more details on Bayesian networks and dynamic Bayesian networks refer to the books [RN06][RN09] and to chapter I sections 2.2.1-2.2.4.

Numerous projects have used dynamic Bayesian networks for context prediction. For example, in Petzold et al. [PP05], the authors used dynamic Bayesian networks to predict a person's indoor movement. Context was represented as DBN, where the current room depends on several rooms visited previously and the duration of staying in the current room depends on the current room, time of the day and day of the week. The context predictor achieved high prediction accuracy (floating between around 70 per cent and around 90 per cent for different persons and tasks). However, retraining the system in case of user habit change turned out to be cumbersome. Dynamic Bayesian networks were also widely used to recognise user plans and infer user goals. The example of user modelling and user goals inference using a Bayesian network is the work by Albrecht et al. [AZ98]. The authors used DBNs to predict further user actions in an adventure game. This task can still be viewed as context prediction – the fact that context is completely virtual does not really affect the essense of the method. Horvits et al., 1998 describes another example of DBNs – the Lumiere project. The project intended to predict the goals of software users and provide services proactively. Nodes of DBN were user profile, goals and actions.

Numerous projects used Bayesian networks for context prediction. Bayesian networks have a broad range of possible use cases and good opportunities to incorporate any kind of prior knowledge.

3.5 Branch Prediction Methods for Context Prediction

Historically, branch prediction algorithms are used in microprocessors to predict the future flow of the program after branch instruction. By their nature, branch prediction methods are fast and simple and designed for fast real-time work. Refer to the work by Yeh and Patt [YP03] for more details on branch prediction techniques.

Petzold et al. [PB03] used branch prediction algorithms for context prediction in pervasive computing systems. Authors used branch prediction to predict moving of the person around the house or office. The system described in [PB03] used several kinds of predictors: state counter-based predictor; state counter-based predictor; local two-level context predictors; and global two-level context predictors. Counter-based predictors were much faster in training and retraining while two-level predictors were much better at learning complicated patterns. The authors also developed some suggestions for enhancements that can take into account time and confidence level.

Context prediction using branch prediction algorithms is not a widespread approach. Algorithms are generally very simple and fast, but they can detect only very simple behaviour patterns.

3.6 Trajectory Prolongation Approach for Context Prediction

Context prediction by trajectory prolongation works in the following manner:

1. Consider every context feature as an axis and construct multidimensional space of context features.
2. Consider the observed context features at a certain moment in time as a point in that multidimensional space.
3. Consider the set of those points collected over time as a trajectory in multidimensional space.
4. Interpolate or approximate that trajectory with some function.
5. Extrapolate that function further in time to get the prediction results.

Some projects used this approach to context prediction. For example, Anagnostopoulos et al., [AM05] suggest a special architecture and approach for context prediction. The

authors validate their approach on location prediction methods for longitude and latitude. For validating and testing, the authors used GPS trace files. According to the authors, Newton and Lagrange's interpolation methods proved to be inappropriate for that purpose due to oscillatory behaviour on a high amount of points. Cubic Bezier splines in turn appeared to be promising and moreover had the complexity only O(n). Regression-based techniques fall under that category as well. For example, Karbassi and Barth [KB03] addressed vehicle time of arrival estimation. The system built linear regression between a congestion factor and time of arrival and inferred the parameters from the history. Some works [SH06][Ma04a] suggested use of autoregressive models to resolve context prediction task.

Trajectory prolongation approach initially comes from a location prediction area which is adjacent to context prediction and can be viewed as its sub task. However, trajectory prolongation can face numerous shortcomings when applied to a general context prediction task. The most important shortcoming for the trajectory prolongation approach is that it is not capable of handling non-numerical data, which are quite common for pervasive computing environments.

3.7 Expert Systems for Context Prediction

Expert systems theory is a branch of the artificial intelligence area which attempts to imitate the work of a human expert. Usually an expert system represents the regularities in terms of rules. For example, it can look like this: *(t>37°C)&(caugh) -> (ill)*

Which means "if the person has body temperature over 37°C and cough, s/he is ill".

Large number of projects applied expert systems to different fields. For more details on expert systems refer to [GR04]. Expert systems are sometimes used for context inference and context prediction. For example, Hong et al. [HS09] use rule engines for context prediction. The system inferred rules in the runtime to determine user preferences and provide services proactively. For example, having user age, gender and family status, the engine can infer what kind of restaurant a user is likely to look for this evening. It is done in terms of rules like: *(age>23)&(age<32)&(gender=male)&(status=married)→(preference=familyRestaurant)*.

Then, for example, the system can predict that the user is going to the restaurant in the evening, proactively provide the choice and find the route in advance. A context aware system employs decision tree learning and rule learning techniques to mine for user preferences.

The system described in [HS09] used Apriori algorithm to infer association rules. Initially that algorithm was developed to mine the data from database transactions and find sets of features that appear together. The main idea of the algorithm is to find the sub sets of facts of different length incrementally starting from single facts. On every step, bigger sub sets are generated and the sub sets with frequency below some thresholds are dropped. For more details on the algorithm and its enhancements refer to the work by Agrawal and Srikant [AS94].

Williams et al. [WM08] discuss forecasting a person's location using the context prediction approach. The system collects the log of household activities and elicits sequential association rules using a generalised sequential patterns (GSP) technique. See [SA96] for more details on the rule mining method.

In another example [Va08] the author considered mining the rules to learn user habits and implement fuzzy control in smart home environments. The pervasive system learns by example from user actions. In brief, their approach looks like this: All the day is divided

into the set of timeframes. The system detects association rules within the timeframe. Every rule has sensor conditions as an input and triggered actuators as an output. The system also maintains a weight coefficient for every rule: triggering the rule increases the weight, not triggering the rules decrease the weight. Rules with 0 weights are removed. If both the weight and the degree of membership for the rule are high enough, the system starts executing the actions proactively, instead of the user, according to the rule.

One more notable work in the area of rule-based systems is the work by S. W. Loke [Lo04a]. The author addressed the questions of proactivity and the questions of action consequences. The author also proposed to determine whether the action was worth performing by comparing the uncertainty of the context and the severity of the action in the form of a rule (expression (10)).

IF Uncertainty(Context) < U AND Severity(Action) < S THEN DO Action (10)

That work [Lo04a] addresses not just rules for taking actions, but different argumentation techniques to define those rules (based on different sources of knowledge) and to determine what action to take.

To summarize, the approach to context prediction based on expert systems is quite promising. It allows quick and natural integration of prior knowledge, it allows relatively easy integration of adaptation actions, and it can contain learning and self-correcting capabilities.

3.8 Context Prediction Approaches Summary

Table 1 presents the comparison of context prediction approaches according to the criteria identified in section 2. It should be noted that different kinds of predictors can be combined to improve prediction quality, enhance each others' strength and compensate each others' weaknesses.

4 General Approaches to Context Prediction

One of the context prediction research challenges is the development of a general approach to context prediction. Many context prediction approaches were designed to fit the particular task and most of them were not designed to be generalizeable (although some of them have generalization capability).

A quite notable attempt to look at the context prediction problem in general was made by R. Mayrhofer who developed a task-independent architecture for context prediction [Ma04a].

As a result, the context prediction process consists of several steps:

1. Sensor data acquisition. This step takes data received from multiple sensors and arranges them into the vector of values.
2. Feature extraction. This step transforms raw sensor data for further usage. From vector of sensor data, vector of features is formed.
3. Classification. Performs searches for recurring patterns in context feature space. Growing neural gas was considered to be the best choice. See [Fr95] for more details on that algorithm. The result of the classification step is a vector of values that represents degrees of membership of current vector to certain class.
4. Labeling. This is the only step that involves direct user interaction. The frequency of involvement depends on a quality of clustering step if classes are often overwritten and replaced that will result in more frequent user involvements.

5. Prediction. This step takes the history of class vectors and estimates a future expected class membership vector. For this step the author researched numerous prediction approaches and, according to evaluation results presented in [Ma04a], an active LeZi – combined with a duration predictor – slightly outperformed other evaluated algorithms.

Many context prediction applications provided the architecture for some particular context prediction tasks. The work by Mayrhofer [Ma04a] was one of the first works which addressed the context prediction task in a general sense and provided complete architecture to handle that challenge. The architecture is well-developed and well-reasoned and it ensures pluggability of different context prediction methods and non-obtrusiveness for the user. The author put large research efforts into estimating the effectiveness of different context prediction approaches and provided quite comprehensive overview of those. The shortcoming here is that the problem of acting on predicted context was not recognised as a challenge and, moreover, using the rules to act on predicted context was considered the only option without any reasoning. That drawback is quite common throughout many general case context predictors. One more minor shortcoming is that feature extraction was restricted only to clustering of sensor data and no other preprocessing was considered.

Several other works addressed general case of context prediction as well. For example, Nurmi et al. [NM05] developed their architecture for context prediction for a MobiLife project. The authors suggested that the process of context prediction consists of several steps: data acquisition, preprocessing, feature extraction, classification and prediction. Actually, the view is quite similar to the one provided in [Ma04a]. However, there are some differences. One of them is that in [NM05] the authors introduced a preprocessing step. As the authors noticed, in [Ma04a] some preprocessing was included into the sensor data acquisition step. However, separating preprocessing and data acquisition can provide more insightful view on the system and make it more flexible. Also Nurmi et al. [NM05] included a labeling step with the classification step and provided some techniques to make labeling even less obtrusive. The shortcomings are generally the same compared to [Ma04a]. The problem of acting on predicted context was slightly mentioned, but no exact solution was proposed.

One more notable work on context prediction architecture is the Foxtrot framework described by Sigg et al. [SH06]. There, authors presented a quite different view of the context prediction problem. Authors focused their efforts on treating the context as time series and applying time series forecasting techniques such as Markov predictors, an autoregressive moving average model or alignment methods. The Foxtrot project represented the context as a multilayered structure where higher level context information was obtained from lower level context information using preprocessing. Context prediction worked on every context layer and provided the prediction for every layer. The approach itself was quite novel. The authors made an extensive research to theoretically estimate the possible error of such an approach. Also, the authors defined a context prediction task in a very general sense and did not make any unnecessary assumptions (like restricting preprocessing to clustering or having only high level context features and low level context data), therefore implicitly assuming only two context layers. The Foxtrot framework implemented prediction on every layer of context data and did not restrict it to higher level context only. However, the choice of possible context prediction methods was relatively small (just autoregressive methods, alignment methods or Markov predictor) and the architecture itself took almost no context prediction specifics into account. The problem of how to use predicted information (including the questions of labeling context classes to achieve meaningful and understandable output of predicted information and the questions of acting on predicted context) was not considered at all.

Anagnostopoulos et al. [AM05] present one more general approach to context prediction. They treat context data as time series, interpolate the trajectory and extrapolate it further for prediction. However, although stated as a general context prediction method, the approach was derived from location prediction techniques validated on location prediction problems and does not take into account general context prediction specifics. Prediction methods were restricted to interpolation (which generally has numerous shortcomings, see sections 3.6 and 3.8), the questions of preprocessing and different layers of context were not taken into account, and the questions of meaningful output or acting on predicted context were not considered as well.

Context prediction and proactivity problems were also addressed in the works regarding context spaces theory. The overview of theory of context spaces can be found in [PL08a]. The theory itself represents the context as a multidimensional space and uses geometrical metaphors to improve context awareness. The situations are represented as subspaces. Context spaces theory was used as a basis for some context prediction and proactive decision-making mechanisms. Padovitz et al. [PL08a] discuss the questions of proactive behavior of a mobile reasoning agent that migrates between different information sources to collect additional information and reason about the situations. In [PL07] the authors provide the concept of natural flow – pre-defined likely sequence of situations over time. Natural flow was used as a verification technique if there were uncertainty about the current situation. Situations that fit the flow are considered to be more likely. Although not used for context prediction directly, the natural flow concept has definite context prediction potential. One more work regarding context prediction and proactivity in context space theory is the work of Boytsov et al. [BZ09]. There the authors provide the techniques to adopt different machine learning based context prediction techniques to context spaces theory.

To summarize, there do exist some projects that address the problem of context prediction in a general sense. However, their number is small and still the development of a general approach to context prediction task is a challenge. The most serious common shortcoming is that acting on predicted context is not recognized as a problem and either is not mentioned at all or considered as a task subsequent to the context prediction step. Meanwhile, in many cases, system actions can influence prediction results and therefore treating context prediction and acting on predicting as independent sequential tasks can severely limit the scope of possible use cases.

5 Research Challenges of Context Prediction

Context prediction is a relatively new problem for computer science. Now the area of context prediction is just being developed and still there are numerous challenges yet to be addressed. Those challenges include:

1. Lack of general approaches to the context prediction problem.
   Most current solutions predict context for particular situations. There have been only a few attempts to define and solve the context prediction task in general.
2. Lack of automated decision-making approaches.
   Most context prediction-related works focused the efforts on prediction itself, but proper acting on prediction results usually was not considered. Most context prediction systems employed an expert system with pre-defined rules to define the actions based on prediction results. With one notable exception of Markov decision processes, almost no systems considered a problem like “learning to act”.
3. Mutual dependency between system actions and prediction results is not resolved.

This challenge is somewhat related to the previous one. Many context prediction systems considered the tasks of predicting the context and acting on predicted context in sequence: predict and then act on prediction results. That approach can handle only simplified use cases when actions do not affect prediction results. For example, in a smart home the system can employ any policy for switching the light or opening the door in advance, depending on user movement prediction results. But whatever the system does, it will not affect user intentions to go to a particular room. However, in a general case system, actions do affect prediction results. For example, consider which is capable of automatic purchases to some degree and which needs to plan the expenses, or in a more serious use case, consider a pervasive system that is capable of calling the ambulance and that needs to decide whether to do it or not depending on observed user conditions. In those and many more use cases, prediction results clearly will depend on what the system does. However, there are almost no works which considered the problem of mutual dependency between system actions and prediction results. So far, the only works which did address that problem were the works on the Markov decision processes (see section 2.3). The task of resolving that dependency is actually a special case of a reinforcement learning task. In our opinion, although comparing to most reinforcement learning task pervasive computing systems have their own specifics (e.g., relatively obscure cost and reward functions, high cost of errors and therefore very limited exploration capabilities), recent advancement in the reinforcement learning area can help to overcome that problem.

If all those context prediction challenges are resolved, it will let pervasive computing systems handle more sophisticated use cases, enhance the applicability and the effectiveness of context prediction techniques and therefore enhance overall usability of pervasive computing systems.

Table 1. An overview of context prediction approaches.[10]

Approach	Prior Knowledge Inference	Run-time Knowledge Inference	Reliability Estimation	Outlier Sensitivity	Observability	Information loss on pre-processing	Data Supported		Memory Amount Needed
							Numeric	Non-numeric	
Pattern Matching	Yes	Yes	Yes	Moderate	White box	Yes	No	Yes	Variable
Markov Models	Yes	Yes	Yes	Moderate	White box	Yes	No	Yes	Fixed
Bayesian networks	Yes	Yes	Yes	Moderate	White box	Yes (usually)	Yes	Yes	Fixed
Neural Networks	No	Yes	No	Moderate	Black box	No	Yes	No	Fixed
Branch Predictors	Yes	Yes	Almost no	Moderate	White box	Yes	No	Yes	Fixed
Trajectory Interpolation	No	Yes	No	Moderate or High	Black box	No	Yes	No	Variable
Trajectory Approximation	Yes	Yes	Yes	Moderate	White box	No	Yes	No	Usually fixed
Expert Systems	Yes	Yes	Yes	Moderate (practically – low)	White box	Yes	No	Yes	Variable

[10] Compared to the article [BZ10a], Table 1 is amended due to space requirements. In *Data Supported* column *Integer* and *Real* data types are merged into *Numeric*. The data in both columns were similar and, therefore, they were merged without being altered.

Chapter IX

Extending Context Spaces Theory by Predicting Run-time Context

Based on:

1. Boytsov, A., Zaslavsky, A. and Synnes, K. Extending Context Spaces Theory by Predicting Run-Time Context, in *Proceedings of the 9th International Conference on Smart Spaces and Next Generation Wired/Wireless Networking and Second Conference on Smart Spaces*. St. Petersburg, Russia: Springer-Verlag, 2009, pp. 8-21.

Foreword

Previous chapter provided an overview of context and situation prediction research area. This chapter continues addressing the research question 3 and improves situation awareness and context prediction techniques by combining versatile situation awareness tools of context spaces approach with various context prediction mechanisms, identified in the previous chapter. Chapter IX is based on the paper [BZ09]. By the time of writing the paper [BZ09] context prediction approaches were considered as a plausible extension of ECORA framework, but eventually became the core of CALCHAS – proactive adaptation framework on top of ECSTRA situation awareness application.

Extending Context Spaces Theory by Predicting Run-time Context

Abstract. Context awareness and prediction are important for pervasive computing systems. The recently developed theory of context spaces addresses problems related to sensors uncertainty and high-level situation reasoning. This paper proposes and discusses componentized context prediction algorithms and thus extends the context spaces theory. This paper focuses on two questions: how to plug-in appropriate context prediction techniques to the context spaces theory and how to estimate the efficiency of those techniques. An overview of existing context prediction methods is presented, including Markov chains, Bayesian reasoning and sequence predictors. The paper also proposes and presents a testbed for testing a variety of context prediction methods. The results and ongoing implementation are also discussed.

Keywords: context awareness, context prediction, context spaces theory, pervasive computing, Markov model, Bayesian network, branch prediction, neural network.

1 Introduction

Pervasive computing is a paradigm where computing systems are integrated into the everyday life and environment in a non-intrusive, graceful and transparent manner. For example, it can be a smart home or office, where doors are automatically opened and light is automatically turned on right before a person enters the room [PB03]. Or it can be a smart car, which suggests the fastest way to the destination and which assesses its own condition and proposes maintenance plan. Many pervasive computing systems are now being introduced into our life.

Context awareness and context prediction are relatively new research areas, but they are becoming an important part of pervasive computing systems. This paper analyzes various context prediction techniques and proposes a plug-in approach to context prediction for pervasive computing systems at run-time. This approach is proposed as an extension of context spaces theory [Pa06][PL04]. This paper focuses on how to apply various available context prediction techniques and how to estimate the efficiency of those techniques. This paper also proposes validation of the pluggable context prediction techniques using the "Moonprobe" model and application scenario that imitates movement of vehicle over a sophisticated landscape.

The article is structured as follows. Necessary definitions are included in section 2. Section 3 provides a brief overview of the context spaces theory – its essence, addressed problems, proposed solutions and current research challenges. In section 4 we propose context prediction methods and develop the algorithms for their adaptation to the theory of context spaces. In section 5 we introduce "Moonprobe" model and application scenario – a testing framework that we developed to estimate context prediction efficiency. Section 6 summarizes the paper and proposes future work.

2 Definitions

This paper addresses many important issues of pervasive computing systems. Generally, pervasive systems comprise many small and inexpensive specialized devices. If a device is used to specifically obtain information from the environment that device is usually referred to as a sensor. Each device performs its own functions, but to be able to perform them effectively the device usually needs to communicate with other devices and process the information that it obtains from them.

One of the most important characteristics of a pervasive computing system is its context. In earlier works on context awareness different definitions of context were proposed. Comprehensive overview of those efforts was presented by Dey and Abowd [DA00]. They define context as "any information that can be used to characterize the situation of an entity." In fact, every piece of information that a system has is a part of that system's context. The system is context aware if it can use context information to its benefit. Reasoning about context is the process of obtaining new knowledge from current and possibly predicted context. Context model is a way of context representation that is used for further reasoning. We assume that context aware pervasive computing systems have the following features:

1. Sensors supply data that will be used by pervasive system for further reasoning. Examples of sensors include light sensors, altitude sensors and accelerometers.
2. Sensors transfer all the data to reasoning engine via a sensor network.
3. The reasoning engine, which can possibly be a distributed system, performs reasoning, makes context predictions and produces some output that can be used for modifying system's behavior.

Many important issues of pervasive computing systems are left out of scope of this paper, for example, security, privacy, need for distributed computations and other related problems.

3 Context Spaces Theory

The theory of context spaces [Pa06][PL04] is a recently developed approach for context awareness and reasoning which addresses the problems of sensors uncertainty and unreliability. It also deals with situation reasoning and the problems of context representation in a structured and meaningful manner.

Context spaces theory is designed to enable context awareness in clear and insightful way. This theory uses spatial metaphors for representing context as a multidimensional space. To understand context spaces theory we need to introduce several new terms. Any kind of data that is used to reason about context is called *context attribute*. Context attribute usually corresponds to a domain of values of interest, which are either measured by sensors directly or calculated from other context attributes. It can be either numerical value or a value from pre-defined set of non-numerical options.

Context state represents the set of all relevant context attributes at a certain time. A set of all possible context states constitutes application space. Therefore, application space can be viewed as a multi-dimensional space where the number of dimensions is equal to the number of context attributes in the context state. The state of the system is represented by a

point in the application space and the behavior of the system is represented by a trajectory moving through the application space over time.

Situation space is meant to represent real life situation. It can be defined as subspace of the application space. So if context state is in the subspace representing situation S, it means that situation S is occurring. Situation S has the level of occurrence certainty. It depends on the probability of the context state to be within S and it also depends on the subspace within S. See figure 1 for simple illustration. We'll keep referring to that example for context prediction approaches illustration throughout the article.

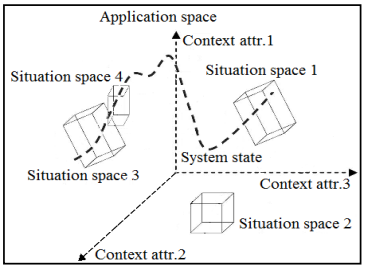


Fig. 1. Context spaces theory

In context spaces theory several methods were developed for reasoning about the context. Bayesian reasoning [RN06] or Dempster-Shafer algorithm [Sh76] are used to get overall confidence in the fact that a situation is occurring. Algebraic operations on situations and some logic-based methods were developed for reasoning in terms of situations.

Context spaces theory was implemented in ECORA [PL08b] – Extensible Context Oriented Reasoning Architecture. ECORA is a framework for development of context-aware applications. That framework provides its functionality as a set of Java classes to be integrated into the prototypes of context-aware systems.

The research presented in this paper develops context prediction methods that can be used to extend the context spaces theory and benefit ECORA-based applications.

4 Context Prediction for Context Spaces Theory

The ability to predict future context will benefit runtime performance of context aware applications. For example, use cases for context prediction can be found in [NM05]. Predicted context can be used, for example, for early warnings, for power management, for system reconfiguration. Context prediction is one of research challenges for context spaces theory.

In practice, most of current context prediction approaches involve inferring various context features during run-time. The results of inference are then used for context prediction. That inference is actually a machine learning task. For this purpose neural networks [Ha99], Markov chains [CM05] and Bayesian networks [RN06] might be sufficiently effective tools. Here we analyze possible context prediction techniques and develop the appropriate algorithms for two-way mapping between the context spaces theory and the analyzed context prediction technique.

Sequence predictors for context prediction. Many context prediction approaches were inspired by research in UNIX next command prediction [DH97][DH98]. There authors suggested to represent each UNIX command as a symbol and the whole command flow – as symbol sequence. Also they proposed a set of algorithms for sequence prediction. Sequence prediction techniques were later used in a variety of systems from "Smart Home" architectures to prediction of movement in cellular networks. Various modifications of LZ78 algorithm [ZL78] were used in various papers [AS02][BD99][DC02][GC03][MR04][RD03]. It is worth noting the paper [MR04] which presents general application-independent architecture for context prediction. The paper [MR04] also suggests that Active LeZi algorithm can be used to predict future situations (note that the concept of a situation in [MR04] differs from the concept of a situation in context spaces theory). The Active LeZi algorithm itself is described in [GC03]. According to [MR04], Active LeZi provides up to 99% of accuracy on 1-step prediction.

Sequence prediction techniques are feasible in context spaces theory as well. For prediction purpose context can be represented as a sequence of situations. When situation occurs, a special symbol is added to the sequence. When situation is over, another special symbol can be added to the sequence as well. Sequence prediction techniques can be applied to predict the next symbol in that sequence and, therefore, to predict the occurrence of a situation. For example, context state flow in Fig. 1 can be described by following sequence: (SS3-in)(SS4-in)(SS3-out)(SS4-out)(SS1-in)

Sequence prediction algorithms should be adjusted to the fact that situations have certain levels of confidence. It can be done by introducing the threshold for the level of certainty. We consider that the situation is occurring if its level of confidence is above the threshold and we consider that the situation is over if its level of confidence drops below the threshold. In [MR04] active LeZi showed very good performance in a variety of cases. Incorporation of active LeZi algorithm into context spaces theory is in progress. Its efficiency is currently being investigated.

Neural networks for context prediction. Some techniques of context prediction use neural networks to learn from user behavior. For example, [Mo98] describes "Neural Network House" project where special equipment predicts expected occupancy patterns of rooms in the house, estimates hot water usag and estimates the likelihood for a room to be entered soon. In [Mo98] authors used feedforward neural networks trained with back propagation. Projects related to indoor movement prediction considered neural network approach as well. In [VG04] authors addressed the problem of predicting next room the person is going to visit. A neural network took current room number as an input and produced the most probable next room number as an output. Logs of user movement were used to train the network. For prediction purposes authors chose multi-layer perceptron with one hidden layer and back propagation learning algorithm.

Neural networks adaptation for context spaces theory can use several approaches. For example, the neural network can accept current situation (with its level of confidence) and predict next situation to occur. Alternatively, the neural network can accept the whole set of current situations and return a whole set of situations expected over a period of time. Additionally, neural network can accept context coordinates and infer the expected future coordinates or expected future situations.

Markov models for context prediction. Markov models proved to be a feasible way of context prediction. In [AZ99][Be96] authors used Markov chains to predict what internet site user is likely to visit next. Context prediction using Markov chains based techniques and Bayesian networks based techniques were considered in the [KL02]. Authors addressed the active device resolution problem, when the user has a remote control to interact with a

set of devices, but there are always several devices in user proximity. Some papers use hidden Markov model to learn from user behavior. For example, in [GV06] hidden Markov models are used to obtain prediction of next room that the person is likely to visit. The layout of rooms in the house, represented as a graph, is handled as a hidden Markov model, and the history is used to obtain the probabilities of moving from one room to another.

Context in our approach can be viewed as a Markov model as well. For that we need to represent the context as a graph where every node represents a situation. However, Markov model states have to be mutually exclusive. It is not generally so in context spaces theory. To make situations mutually exclusive, we need to find all kinds of situations intersection and extract them as new situations. To do this we developed the following algorithm.

Algorithm MM4CS.

Step 1. Create three entities:

Entity 1: set of situations. Initially it contains all defined situations. From now on initial situations will be referred to as S_1, S_2, S_3, …, S_N. Also they will be referred to as old situation spaces. Newly acquired situations may appear as a result of situation decomposition. They will be referred to according to the situation algebra expressions they were obtained from, for example, $S_1 \cap S_2 \cap !S_3$ or $S_{10} \cap S_8$. From now on, for simplicity, when talking about situations and it does not matter whether those situations are initial or newly acquired we'll refer to them as A,B,C,etc. Subspace of application space that includes whole application space except the situation A will be referred to as !A.

Entity 2: set of situation pairs. Elements of the set will be referred like (A, B), where A and B are some situations. Expression (A,*) is a joint name of all elements containing the situation A. Pair of situation with itself like (A,A) are not supported. Situation pairs are unordered, i.e. (A,B) and (B,A) both identify the same set element. Initially the set contains all available pairs of starting set of situations, i.e. (S_i, S_j) for every $i,j \in [1..N]$ and $i \neq j$.

Entity 3: dependency graph. Initially it has 2 sets of vertices. Each vertex of each set represents a situation. Vertex will be referred as $(A)^j$ where the A represents situation and the superscript represents the set number (1 or 2). Dependency graph will be used for prediction in terms of old situation spaces. Initially it has only vertices $(S_i)^1$ and $(S_i)^2$ and edges between $(S_i)^1$ and $(S_i)^2$ ($i \in [1..N]$). Edges are undirected. Also in the graph special temporary vertices can appear during the runtime. They will be referred as $(A)^{j:B}$.

Step 2. Take any element from a situation pairs set. Let's say, it is element (A, B). See, if situation spaces A and B have any intersections in application space. It can be done using situation algebra provided in [Pa06]. If there are no intersections between A and B – just remove the element (A, B) from the situation pair set. If there are any intersections, a set of substeps has to be performed:

Substep 2.1. Remove element (A, B) from the set.

Substep 2.2. Iterate through the situation pairs list and perform following changes:

Substep 2.2.1. Every (A,*) element should be replaced with two new elements. Element (A∩B, *) is created in all cases and element (A∩!B, *) is created if the subspace A∩!B is not empty.

Substep 2.2.2. Every (B,*) element should be replaced with two new elements as well. Element (A∩B, *) is created if it was not created on substep 2.2.1 and element (!A∩B, *) is created if the subspace !A∩B is not empty.

Substep 2.3. Update dependency graph:

Substep 2.3.1. Vertex $(A)^2$ should be removed and several new vertices should be created instead. Vertex $(A \cap B)^{2:A}$ should be created in all cases and vertex $(A \cap !B)^2$ should be created if subspace A∩!B is not empty. Newly created vertices should have the edges to every element that $(A)^2$ had the edge to.

Substep 2.3.2. Vertex $(B)^2$ should be removed and several new vertices should be created instead. Vertex $(A \cap B)^{2:B}$ should be created in all cases and vertex $(!A \cap B)^2$ should be created if subspace $!A \cap B$ is not empty. Newly created vertices should have the edges to every element that $(B)^2$ had the edge to.

Substep 2.3.3. Vertices $(A \cap B)^{2:A}$ and $(A \cap B)^{2:B}$ should be merged to new vertex $(A \cap B)^2$. All the edges leading to either $(A \cap B)^{2:A}$ or $(A \cap B)^{2:B}$ should be redirected to $(A \cap B)^2$ instead.

Substep 2.4. In situation set remove situations A and B and introduce all non-empty situations of these: $A \cap B$, $!A \cap B$, $A \cap !B$.

Step 3. If situation pair list is not empty – go to step 2. If situation pair list is empty – add new situation to the situations list: $!S_1 \cap !S_2 \cap !S_3 \cap !... \cap !S_N$. Also add corresponding vertex to the dependency graph: $(!S_1 \cap !\ S_2 \cap !S_3 \cap !... \cap !S_N)^2$. It has no edges. Another option is not to mention $!S_1 \cap !\ S_2 \cap !S_3 \cap !... \cap !S_N$ situation and consider the system being in process of transition when no situation is triggered. ■

Resulting list of situations will now be referred to as new situation spaces. Resulting dependency graph will have old situation spaces in one set of vertices and new situation spaces in another. Edges between the vertices mean that old situation space has corresponding new situation space as a part of its decomposition.

Algorithm has to be applied to the system only once – before the runtime. When this algorithm is complete, mutually exclusive set of situations is obtained. Now this new set of situation spaces can be represented as states of the Markov model. All Markov model based context predictors are applicable for it.

For illustration see Fig. 2. There are two options of Markov models for the system described in Fig. 1 and it depends on what option do we take on step 3 of an algorithm.

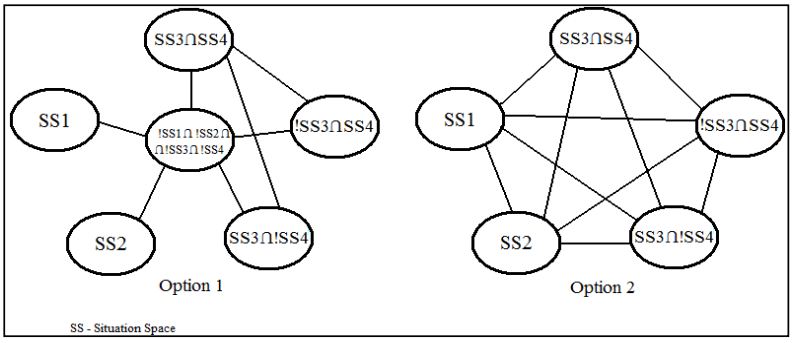


Fig. 2. Markov model for Fig.1

Reasoning in terms of old situation spaces can be easily done using new situation spaces. Let's say, we need prediction for the situation S_i from old situation spaces. It can be done in the following manner.

Step 1. Find $(S_i)^1$ vertex. It is connected to some other vertices. Let them be $(D_1)^2$, $(D_2)^2$, $(D_3)^2$, … $(D_k)^2$,etc.

Step 2. Take predicted levels of confidence for those vertices. It is done using Markov predictor directly.

Step 3. Infer the level of confidence of situation:

$P(S_i) = P(D_1 \cup D_2 \cup D_3 \cup \ldots \cup D_k)$

Where $P(S_i)$ – predicted level of confidence in situation S_i, and sign U is a operator for uniting situations. Uniting situations can be done by the means of situation algebra provided by [Pa06]. $P(S_i)$ is the demanded result. However, these 3 steps should be done on every case of prediction and therefore they will introduce some run-time overhead.

So, Markov models seem to be feasible way of context prediction for context spaces theory. However, to make it applicable for context spaces some pre-processing is needed. Some computation overhead in the run-time will be introduced as well to transition from new situation spaces to old ones.

Bayesian networks for context prediction. Bayesian network approach for context prediction was considered in the paper [PP05]. That research addressed the problem of predicting next room person is likely to visit. Context was represented as dynamic Bayesian network, where current room depended on several rooms visited previously and duration of staying in current room depended on current room itself, time of the day and week day. In [PP05] dynamic Bayesian network learned from the history of indoor movements. One more case of context prediction using Bayesian networks was described in [KL02]. There authors addressed active device resolution problem. The paper [KL02] was already discussed in previous section.

Context within context spaces theory can actually be represented by dynamic Bayesian network. In a simplest form it can allow to establish correlations between situations in time (and situations do not have to be mutually exclusive in this case). Dynamic Bayesian networks can also help if there are any additional suggestions about factors that influence the context but are not included directly in the application space.

Branch prediction for context prediction. Paper [PB03] considered applying branch prediction algorithms for context prediction in pervasive computing systems. Branch prediction algorithms are used in microprocessors to predict the future flow of the program after branch instruction. In [PB03] the authors consider another example: branch prediction is used to predict moving of the person around the house or office.

Branch prediction algorithms can be applied to context spaces theory as well. Once again, context should be represented in terms of situations, and the situations should be represented as a graph. One option is to use mutually exclusive situation decomposition that was presented for Markov model predictors. Then we can predict moving through the graph over time using branch prediction techniques. Another option is to use the same approach like we took to apply sequence predictors: when situation happens or wears off, special symbol is added to the sequence. Branch predictors can be used to predict next symbol there.

Summary. The features of different approaches to context spaces theory can be summarized in following way (see table 1).

Common challenges and future work directions. As it was shown all context prediction methods have to overcome some specific challenges to be applied to context spaces theory. But there is one common challenge, which was not yet addressed: all the defined context prediction methods deal with a discrete set of possible events or situations. So to apply the prediction methods context should be represented as a set of situations. But the flow of any prediction algorithm depends on how we define situation spaces within application space. Different situation spaces will result in different algorithm flows, even if application space is generally the same. This effect might be vital. So the question arises: what is the best way to define situations inside context space for prediction purposes? Results can depend on exact prediction algorithm or exact application. Situations should be both meaningful for the user and good for prediction. This question needs further research.

Table 1. Context prediction approaches summary

Approach	Benefits and shortcomings	Summary
Sequence predictors	Good results are proven in practice. Low adaptation efforts are needed.	This approach is the most perspective .
Neural networks	Large variety of options to choose and low adaptation efforts needed. However, additional thorough testing is needed by every application to determine what exact neural network type is the most feasible.	The approach is quite perspective.
Markov chains	They can be applied. However, this approach requires splitting the application space in a set of non-overlapping situations, which is not natural for context spaces theory and requires significant pre-processing	The efficiency of this approach is questionable.
Bayesian networks	Approach is able to estimate the influence of different factors, not depending on whether they are in the context or not yet. Low adaptation efforts are needed.	The approach is very perspective.
Branch predictors	The algorithms are simple. However, prediction results are influenced only by small part of the history and the algorithms are capable of working with simple patterns only.	In general case that approach is not perspective.

5 Testbed for Context Prediction Methods

For validating context prediction methods proposed above we introduce "Moonprobe" model and application scenario. "Moonprobe" is a model that simulates vehicle going over some sophisticated landscape in 2D environment. The aim of the vehicle is to reach some destination and not to crash or run out of fuel on its way. This model was developed using XJ Technologies AnyLogic modeling tool [A12].

The "Moonprobe" model deals with the problems that every pervasive system deals with, inclufding sensor uncertainty, sensor outages, determining environment characteristics, fusing heterogeneous sensor data. Also the moonprobe has a set of specific situations to be aware of. These are not just sensor outage situations, but also current progress of going to the destination point and safety conditions. By now the moonprobe uses around 20 context parameters to reason about the situations.

In a very simplified manner the model can be described by the following set of equations (see formula (1)).

$$\begin{cases} \dfrac{d\bar{x}}{dt} = \bar{V} \\ \dfrac{d\bar{V}}{dt} = \dfrac{\mathrm{M}*\bar{\mathrm{g}} + \bar{\mathrm{N}} + \bar{\mathrm{F}}\mathrm{en}}{\mathrm{M}} \\ \bar{\mathrm{N}} = \begin{bmatrix} \text{If probe is in the air: } 0 \\ \text{If probe is on the ground: projection of } [-\mathrm{M} * \bar{\mathrm{g}} - \bar{\mathrm{F}}\mathrm{en}] \\ \text{on the axis perpendicular to the ground} \end{bmatrix} \\ \dfrac{dFl}{dt} = -FCR * |\bar{\mathrm{F}}\mathrm{en}| \end{cases} . \quad (1)$$

Where *t* is time, *X* is probe coordinate, *V* is probe velocity, *M* is mass of the probe (known system parameter), *N* is support reaction (directed perpendicular to the ground), *Fl* – fuel level remained, *g* – gravity, *FCR* – fuel consumption rate (known system parameter), *Fen* – engine force vector (value and direction set by probe).

In some cases probe can leave the ground. When probe lands, it will experience ground collision. Ground collisions are dangerous for the probe. Engines can be used to slow down the landing and avoid the crash.

The architecture of the model is following (see Fig. 3). System consists of several components.

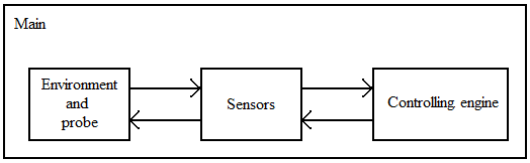


Fig. 3. "Moonprobe" system architecture

1. "Environment and probe" component. It implements the physical model of the system. Also it monitors what happens to the probe (e.g. crash, sensor outage).
1. "Sensors" component. Sensors measure the data from the probe in certain moments of time and introduce some errors.
2. "Controlling engine" component. It takes sensed data and provides control information (e.g. desired engine power).
3. "Main" component provides overall assessment of experimental results.

All the context prediction methods described in section 4 can be applied to moonprobe scenario. They can be incorporated in "Controlling engine" component and then be used to find optimal values of engine power to minimize crash risk and reduce fuel consumption.

The development of the "Moonprobe" testbed is now complete. Screen dump of running "Moonprobe" is depicted on Fig 4.

Model evaluation has shown that the "Moonprobe" model is really capable of estimating prediction results and that context prediction and context awareness algorithms are really pluggable into it. Exact measurements of the result of different context prediction methods are in progress.

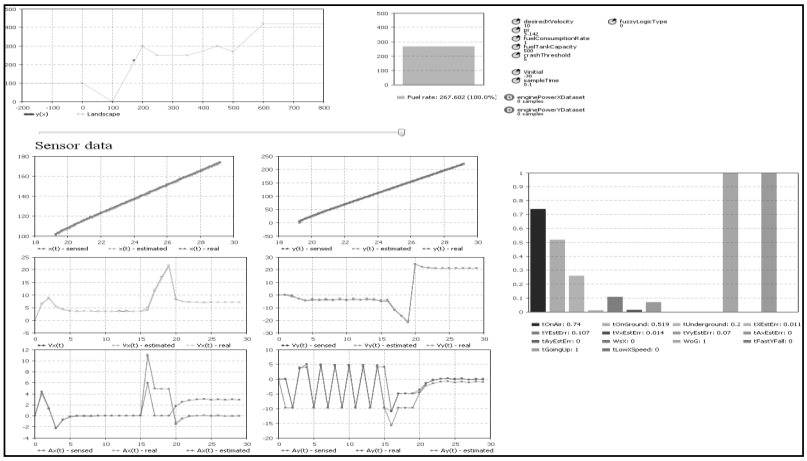


Fig. 4. "Moonprobe" system working

6 Conclusion and Future Work

Context spaces theory is a promising approach, which addresses many problems of context awareness in pervasive computing. Adding accurate and reliable context prediction methods is one of the research challenges of that theory.

In this paper a set of available context prediction techniques was proposed. Two-way mapping algorithms were developed to apply these context prediction methods to context spaces theory. Efficiency of context prediction methods was addressed. A special testbed was developed to evaluate the effectiveness of context prediction methods. Feasibility of the proposed testbed has been also proven.

As a result, context spaces theory was enhanced with context prediction algorithms and their applicability and feasibility were demonstrated. By extending context spaces theory with context prediction techniques it can address not only problems of uncertain sensors or situation reasoning, but context prediction problem as well. It will benefit ECORA-based applications and widen the applicability of ECORA to new class of tasks – the tasks which require prediction capability. The proposed approach incorporates pluggable algorithms of context prediction, so for every application it can be defined what exact context prediction approach is the best.

Future work will concentrate on accuracy metrics and analytical comparison of the efficiency of context prediction methods. Another interesting challenge for future work is how to define the situation spaces in a manner that both grants usability and enhances context prediction.

Chapter X

Extending Context Spaces Theory by Proactive Adaptation

Based on:

1. Boytsov, A. and Zaslavsky, A. Extending Context Spaces Theory by Proactive Adaptation. in Balandin, S., Dunaytsev, R. and Koucheryavy, Y. eds. *Proceedings of the 10th international conference and 3rd international conference on Smart spaces and next generation wired/wireless networking (NEW2AN'10/ruSMART'10)*, Springer Berlin / Heidelberg, 2010, pp. 1-12.
2. Boytsov, A. Proactive Adaptation in Pervasive Computing Systems. in *ICPS '10: Proceedings of the 7th international conference on Pervasive services*, Berlin, Germany: ACM, 2010.

Foreword

Chapters VIII and IX mainly addressed the first part of the research question 3 – how to predict future situations? Those chapters proposed the techniques for situation prediction by combining various predictive models with situation awareness approach of context spaces theory. However, situation prediction should be complemented by properly acting according to prediction results.

As chapter VIII concluded, most current context prediction and situation prediction approaches consider context prediction and proactive adaptation tasks as sequential. This approach does not work if prediction results can be influenced by pervasive system actions. Chapter X conlcudes addressing the research question 3 and answers the second part of the research question 3 of how to properly act according to predicted context. This chapter continues the research started in chapter IX and addresses the challenge identified in chapter VIII. The proposed CALCHAS application is integrated as an extension of ECSTRA framework.

Extending Context Spaces Theory by Proactive Adaptation[11]

Abstract. Context awareness is one of the core features of pervasive computing systems. Pervasive systems can also be improved by smart application of context prediction. This paper addresses subsequent challenge of how to act according to predicted context in order to strengthen the system. Novel reinforcement learning based architecture is proposed to overcome the drawbacks of existing approaches to proactive adaptation. Context spaces theory is used as an example of how existing context awareness systems can be enhanced to achieve proactive adaptation. This recently developed theory addresses problems related to sensors uncertainty and high-level situation reasoning and it can be enhanced to achieve efficient proactive adaptation as well. Possible reinforcement learning solutions for pervasive computing area are elaborated.

Keywords: context awareness, context prediction, context spaces theory, pervasive computing, reinforcement learning, proactive adaptation, act-ahead adaptation.

1 Introduction

Pervasive computing is a paradigm where computing systems are integrated into the everyday life in a non-intrusive, graceful and transparent manner. Some implementations of pervasive computing paradigm include smart homes, elderly care systems, smart mobile devices, GPS navigators, RFID tracking systems, social networks.

Context awareness is one of the basic features of pervasive computing. Context prediction is also recognized as a challenge and an opportunity. However, for context prediction and especially for acting on predicted context there is a definite lack of universal approach to the problem.

This paper proposes and motivates the approach for contextual act-ahead adaptation – proactive adaptation to predicted context. The strengths and challenges of proposed approach are discussed.

The article is structured as follows. Section 2 describes related work and current challenges of proactive adaptation to predicted context. Section 3 further elaborates identified challenges and proposes and explains the solution approach. Section 4 describes the essentials of context spaces theory and introduce ECORA framework. Sections 5 and 6 discuss the integration of proposed approach into context spaces theory and introduce CALCHAS – Context Aware aCt aHead Adaptation System. Section 7 discusses the plausible reinforcement learning methods and their application to the context model. Section 8 makes a summary, provides future research plans and concludes the paper.

[11] The publications [BZ10b] and [Bo10] were merged to create this chapter. References and formulas were renumbered accordingly. Sections 1-6 correspond to sections 1-6 of the publication [BZ10b], while the section 7 corresponds to section 7 of the publication [Bo10].

2 Context Prediction and Acting on Predicted Context

Most of context prediction approaches use machine learning techniques. Predictive models that were applied to context prediction task include sequence predictors [DC02], neural networks [TW08], Bayesian networks [PP05], branch predictors [PB03], decision tree learning [HS09], time series prediction [SH07], Markov models [GV06], trajectory extrapolation [AM05].

Some authors actually do recognize acting on predicted context as a specific problem [CA09][CD07], but still there is not much research done for the challenge of proactive adaptation. Generally most of the ways to act on predicted context can be classified in two groups.

1. Rule-based engines. For every particular prediction result there is a rule that defines an action. Improving behavior of the system is achieved by improving efficiency of context prediction.

2. Learning by example. That approach was applied mostly in smart home environments. Pervasive computing system tracks user actions and then it starts executing the actions for the user.

On the basis of studying and analyzing existing approaches, summary representation can look like expression set (1).

$$\begin{aligned} &state(t)=SenseAndProcessData(t)\\ &prediction(t)=PredictionModel(state(t),history(t),prediction(t-1))\\ &history(t+1)=addToHistory(state(t),history(t))\\ &action(t)=actOnPrediction(state(t),prediction(t),history(t)) . \end{aligned} \quad (1)$$

Where *state(t)* is entire context state at the time t, including the results of sensor data acquisition, validation and processing. Entire aggregated history up to time t, but not including time t, is stored in *history(t)*. Prediction results at time t for time t+1 and maybe subsequent steps are referred to as *prediction(t)*. Usually they depend on the current state and on the dependencies learned from history. Sometimes previous prediction results can influence them as well, if time horizons of prediction attempts do overlap. System is acting on predicted context, so action at time t *action(t)* depends on prediction results and current state. In case learning by example is implemented, action also has some learning mechanisms and therefore depends on history.

The model presented in (1) has a very serious drawback – it cannot handle mutual dependency between system actions and prediction results. Those drawbacks and solution opportunities will be addressed in section 3.

Notable exceptions that do not fall in that model are the applications of Markov decision processes and Q-learning based approaches to context prediction [FL07][Mo04][ZM08]. They will be mentioned further in section 3 when discussing reinforcement learning solutions.

3 Proactive Adaptation as Reinforcement Learning Task

Sometimes the system has to provide results of context prediction in quite specific conditions. Here are some motivating use cases. Consider elderly care system. Users need some assistance in accomplishing everyday tasks. System is able to give an advice, call a

caregiver or just do nothing if the user is performing quite well. The task is to predict the probability of successful outcome of the activity and to act to maximize that probability, but not at the cost of wasting the time of the caregiver or annoying the user with unnecessary advices. Use case is partially based on [HB07].

Another motivating use case can be a smart home, where user satisfaction can be estimated according to both direct questioning and the number of user manual interventions into a configuration of home parameters. The task is to predict user's cumulative satisfaction and act to increase it. Also the system should do its best to behave in non-intrusive manner and not to annoy the user with too many questions.

One more example might be prediction and prevention of memory shortage in pervasive system. If memory shortage was predicted with the certainty above the threshold, the system takes actions to avoid memory shortage condition (e.g. searches for additional resources to share the work). But if those preventive actions are successful, the predictor would have learning problem. Formally if there was no memory shortage, it is a prediction failure. In reality most likely it is not, but we cannot claim it for sure. Actually, when the system starts decision making, predictor becomes unable to learn properly - predictor cannot distinguish between prediction failure and successful prevention.

For all those cases, neither rule-based system, nor learning by example can solve the problem completely. The solution for this and many similar use cases is to take decision making process into account while making predictions.

The dependencies for those use cases can be described like the expressions (2).

$$
\begin{aligned}
&state(t)=SenseAndProcessData(t)\\
&prediction(t)=PredictionModel(state(t),history(t),prediction(t\text{-}1),\\
&\qquad \boldsymbol{action(t)})\\
&history(t+1)=addToHistory(state(t),history(t))\\
&action(t)=actOnPrediction(state(t),prediction(t),history(t)) .
\end{aligned}
\qquad (2)
$$

The meaning of *state(t)*, *history(t)*, *action(t)* and other elements in (2) is the same as in expressions (1). The difference between (2) and (1) is marked in bold. The additional dependency makes most current proactive adaptations methods inapplicable. System acts on predicted context, but in turn predicted context depends on system actions. Most of the context prediction approaches we mentioned previously use predictive models, which do not take into account the mutual dependency between system actions and prediction results.

Usually this problem is avoided by completely splitting the context in two independent parts: the one that is affected by actions and the one that is predicted. For example, if the system intends to proactively open the door before the user comes into the room, the choice whether the system opens the door or leaves it closed will not change user intentions and, therefore, prediction results.

Learning by example is also the way to avoid mutual dependency problem, but this approach has very limited applicability: it works only if possible user actions and system actions significantly overlap and only with the assumption that imitating user actions grants acceptable effectiveness of the system.

We propose an enhanced method to solve mutual dependency problem between actions and predictions. That problem can actually be viewed as reinforcement learning task. Reinforcement learning problem is a problem faced by an agent that should find an acceptable behavior in a dynamic environment and learn from its trial and errors [KL96]. Reinforcement learning in application to pervasive computing task is almost not researched. Notable exceptions are Markov decision processes and Q-learning approaches [FL07][Mo04][ZM08]. However, as it will be discussed further in this section, Markov

decision processes have very limited applicability due to the features of pervasive computing systems. Recent advancements in reinforcement learning include the approaches, which to the best of our knowledge were not applied to pervasive computing at all, including predictive state representation [WJ08], most cases of neural network control [HD99], different applicable cases of actor-critic model (like [HW07]), self-organizing maps and Q-learning integration [S02] and many more.

To the best of our knowledge, there was no attempt to address particular features of reinforcement learning in pervasive computing area. Those features are:

1. Continuous action spaces. Actuators sometimes provide the choice from practically continuous range of values.
2. Continuous state spaces. Many kinds of sensors produce values from practically continuous range as well. However, this problem can be resolved by the means of situation awareness.
3. Mobility of nodes. Mobility of nodes in pervasive computing system can cause the loss of connection to certain sensors and actuators. Or, on the contrary, new connections to different sensors and actuators can be established.
4. High dimensionality. The count of sensors and actuators in pervasive computing system can be very high.
5. High heterogeneity of data. Discrete-valued, continuous-valued or event-based sensors and actuators can appear in any combination.
6. Ability to incorporate prior knowledge. Sometimes common sense or expert estimations can give some sketches of good acting strategies. Ability to incorporate them can significantly reduce learning time.
7. Explicit prediction result can also provide some insight into the problem.
8. Limited exploration capabilities. Pervasive system cannot just do a random thing to see what happens.
9. Limited time for decision making.
10. Goals of the user can change instantly.

The features mentioned above seriously limit the scope of reinforcement learning solutions that we can try. In particular, those features mean that Markov decision process – the dominating model for reinforcement learning – can be used only in limited set of special cases. The main reason here is MDPs' discrete state and action spaces, which might be not suitable for pervasive computing solutions.

In sections 4-7 we will discuss in more details, how we can address pervasive computing challenges presented in the list above.

4 Context Spaces Theory – Main Concepts

Some features of proactive adaptation in pervasive computing can be addressed by improving context spaces theory. The theory of context spaces [PL04] is a recently developed approach for context awareness and reasoning which addresses the problems of sensors uncertainty and unreliability. It also deals with situation reasoning and the problems of context representation in a structured and meaningful manner.

Context spaces theory is designed to enable context awareness in clear and insightful way. This theory uses spatial metaphors for representing context as a multidimensional

space. To understand context spaces theory we need to introduce several new terms. Any kind of data that is used to reason about context is called *context attribute*. Context attribute usually corresponds to a domain of values of interest, which are either measured by sensors directly or calculated from other context attributes. It can be either numerical value or a value from pre-defined set of non-numerical options.

Context state represents the set of all relevant context attributes at a certain time. A set of all possible context states constitutes application space. Therefore, application space can be viewed as a multi-dimensional space where the number of dimensions is equal to the number of context attributes in the context state. The state of the system is represented by a point in the application space and the behavior of the system is represented by a trajectory moving through the application space over time.

Situation space is meant to represent real life situation. It can be defined as subspace of the application space. So if context state is in the subspace representing situation S, it means that situation S is occurring. Situation S has the level of occurrence certainty. It depends on the probability of the context state to be within S and it also depends on the subspace within S. See Figure 1 of chapter IX for simple illustration.

In context spaces theory several methods were developed for reasoning about the context. Bayesian reasoning [RN06] or Dempster-Shafer algorithm [Sh76] are used to get overall confidence in the fact that a situation is occurring. Algebraic operations on situations and some logic-based methods were developed for reasoning in terms of situations [PL04].

Some solutions were developed to integrate various context prediction methods into the theory of context spaces [BZ09]. However, those prediction methods are based on pure forecasting and do not take into account decision making aspects.

Context spaces theory was implemented in ECORA [PL08b] – Extensible Context Oriented Reasoning Architecture. ECORA is a framework for development of context-aware applications. That framework provides its functionality as a set of Java classes to be integrated into the prototypes of context-aware systems.

The presented research introduces proactive adaptation methods that can be used to extend the context spaces theory and enhance ECORA-based applications.

5 Integrating Proactive Adaptation into Context Spaces Theory

Context spaces theory has all the necessary capabilities to ensure context awareness. However, when it comes to proactive adaptation, some enhancements have to be made. At first, the actuators need to be introduced into the model. Actually, those actuators can constitute a separate space in the manner much like the context space itself. That actuator space can be treated as action space for reinforcement learning. Considering different meaning of sensors and actuators, it seems that the better solution is to separate the spaces for sensors and actuators. Situation spaces in the space of actuators are not likely to have any value (or, in case a set of discrete actions can be elicited, those situations will just look like points, not spaces).

In case of the lost connection to actuator, the corresponding axis is removed. Most likely, it will take some time for the system to learn how to act in new conditions, but previously learnt data can provide a good starting point and reduce adaptation time. Similar

refers to introducing new actuators – combining old data with exploration can provide adaptation to new capabilities.

Another question is how to define reinforcement learning state space for context spaces theory. Having discrete state space is likely to make the task simpler. One solution is to take situations as states for reinforcement learning and trigger the state if the certainty of situation occurring is above some threshold. To fit the Markov property (that is essential for different kinds of Markov decision processes), situation decomposition algorithm presented in [BZ09] can be used. That kind of situation decomposition approach can also help to overcome node mobility problem from sensor point of view: if the sensor loses the link or if new sensor is introduced, situation properties are recalculated (see [PL04] for the details), but it is transparent for the upper layer, including the proactive adaptation.

Also taking context space as state space for reinforcement learning is an option. One more option is to take the degrees of certainty in different situations as continuous state space.

To summarize, context spaces theory can be enhanced to incorporate proactive adaptation solutions. Those context spaces theory enhancements can deal with one of the main features of pervasive computing – the mobility of sensors and actuators – using geometrical metaphors and situation awareness. Addressing other features of proactive adaptation depends mostly on exact reinforcement learning method and the architecture of proactive adaptation solutions of upper layers.

6 CALCHAS Prototype

The proposed approach can be implemented by creating a general-purpose middleware for context prediction and proactive adaptation based on context spaces theory. CALCHAS (Context Aware Long-term aCt aHead Adaptation System) prototype is now under development. Integrated with ECORA, it will allow achieving efficient proactive adaptation for context spaces theory.

The architecture was designed to address all the features of pervasive computing area that were mentioned in section 3.

That middleware solution should incorporate a library of different context prediction and proactive adaptation methods. Middleware development is in progress and the prototype is being implemented.

We propose the following architecture for CALCHAS (see Figure 1).

The architectural blocks have following purpose. Run-time goal and efficiency engine translates user input into exact goals for pervasive system in terms of context and timing. Sensor fusion and low-level validation block provides preliminary low-level processing of context data. Retraining database provides bulk of recent low-level data on request, in case there is a need to retrain the model (e.g. due to goal change) or provide faster model start. Feedback is implicit – every actuator action will affect future state of the system, which will in turn be tracked by sensors. Adaptation and prediction engine is responsible for inferring adaptation sequences – sequences of actions for the actuator to do. Also it is responsible for providing explicit prediction results. Adaptation engine has the following structure (see Figure 2).

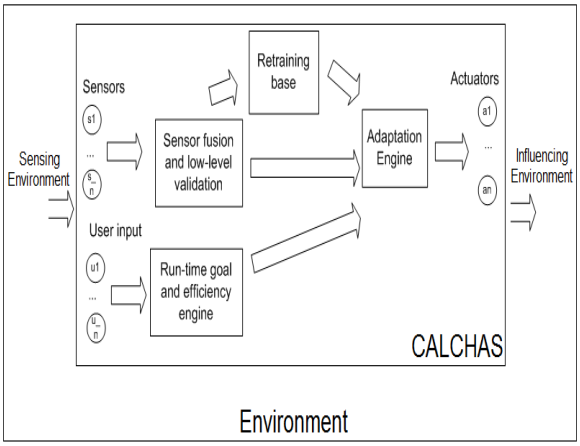


Fig. 1. CALCHAS general architecture.

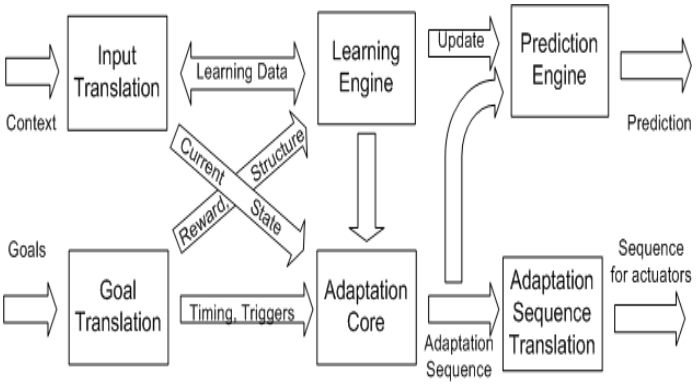


Fig. 2. CALCHAS adaptation engine.

All the translation blocks are used to translate between the format of internal model (e.g. states of Markov decision process, its reward functions) and the format of the outer system (e.g. vectors of values of sensor data and exact commands for actuators). The adaptation core is responsible for generating the actions according to the model and learning engine is responsible to adapt the model to new data.

All translation blocks are heavily dependent on specific task, but for most of other blocks general purpose solutions can be developed. Supplied with a comprehensive library of reinforcement learning methods, the system can facilitate the development of pervasive computing applications, which will be aware of current and predicted context and which will be capable of smart proactive adaptation to predicted context.

7 Reinforcement Learning Solutions[12]

Now we can estimate what kind of reinforcement learning solutions can be plugged into CALCHAS. Features of pervasive computing, identified in section 3, should be also taken into account.

To discuss reinforcement learning approaches we'll need to introduce some definitions. Refer to [SB98] for general reinforcement learning overview.

Reinforcement learning problem is the problem of an agent that acts in unknown environment. It can execute the actions and receive the observations (e.g. new state, immediate reward or cost). The majority of reinforcement learning solutions were designed for the case when the environment can be modeled by Markov decision process (MDP). A particular finite MDP is defined by its set of states S, action set A, one-step dynamics of the environment $P(S_{t+1} \mid S_t, a)$ and cost/reward function R(s) (sometimes – R(s,a) or even $R(s_t,a,s_{t+1})$). From here and now we consider that the time in the model is discrete.

The method to choose the actions depending on a state is usually referred to as a policy. The policy might as well be stochastic. Usually it is considered that the optimal policy is the policy that maximizes expected discounted sum of rewards. The state value function of the state $V_\pi(s)$ is the expectation of future sum of rewards, if in state *s* we follow policy π. In turn, V(s) is value function of an optimal policy. Action-value function $Q_\pi(s,a)$ of the state *s* and action *a* means the expected future reward of taking action *a* in state *s* and following policy π afterwards. Action-value function Q(s,a) of the state s and action a means the system takes action a in a state s and follows optimal policy π after. If action-value function is known, the optimal action for state s will be $\text{argmax}_a(Q(s,a))$. However, in practice Q-function is usually unknown, and the strategies are more complicated.

In contrast to the model defined above, in pervasive computing the sensors and actuators usually have practically continuous range of available values and discretization is not always possible. So, continuous ranges of state and actions fit the features of pervasive computing area in a better way. All the concepts defined above can be generalized for continuous range of state and actions in a straightforward manner. However, it introduces a set of new challenges.

Here we present some attempts of generalization of reinforcement learning solutions to continuous space. Now most of them are under development in CALCHAS system.

7.1 Q-learning in Continuous Space

Q-learning approach is relatively simple, yet very effective way to learn action-value function from interacting with the environment. Initially Q(s,a) can have any values. On subsequent steps the agent updates Q-function in a following way (formula (3))

$$Q(s_t,a_t) \leftarrow (1-\alpha)*Q(s_t,a_t) + \alpha*(R + \gamma*\max Q_t(s_{t+1},a_{t+1})) \qquad (3)^{13}$$

Where γ is discount factor and α – learning rate. See [S02] for more detailed description of the method.

[12] Comparing to the publication [Bo10], numbering of formulas was introduced to this section. The text was amended accordingly.

[13] Original publication [Bo10] contained a typo in the corresponding formula. For clarity, in this chapter the formula is corrected.

In continuous state and action space action-value function is usually approximated. The most serious problem for continuous Q-learning generalization is the choice of actions to take. When the set of available actions is finite, searching for the action with largest Q-value can be straightforward. However, when the actions can be chosen from continuous range, exhaustive search does not work. There were some attempts to avoid that problem and generalize Q-learning approach to continuous state and/or action spaces. For example, those are:

1. **Wire fitting approach.** The approach learns the state-action function Q(s,a) in a format that allows finding the maximum easily. In brief, this method learns a set of functions $A_i(s)$ and $Y_i(s)$, that are referred to as control wires. Action-value function Q(s,a) is approximated in such a way that for every state the maximum of function Q(s,a) is the highest value of a wire function: $\max_i(Y_i(s))$ and corresponding $A_i(s)$ contains the action which leads to $Q(s,a)=Y_i(s)$. The detailed description of the method can be found in [BC93].
2. **Q-AHC.** That method which combines Q-learning with actor-critic learning. Q-learning is used to choose between a set of actor-critic learners. However, the work [GW99] claims that this method did not show good results and tended to set the actions to constant settings or use only one actor-critic model. See [GW99] for more details about the method.
3. **CMAC integration.** Being applied to reinforcement learning, CMAC (Cerebellar Module Articulation Controller) approach divides the entire state and action space into cells by a set of overlapping grids (tilings). Each cell has an associated weight. As the tilings overlap, every point (s,a) in the combined state and action space corresponds to several cells at once. If the system is in state (s,a), those corresponding cells are counted as triggered. Action-value function is approximated by a sum of weights for all triggered cells. On every step the weights of tiles are adjusted according to state, actions and received rewards. See [SS98] for more details on the method. The major drawback for CMAC integration method is that there is no efficient way for action selection in continuous action space.
4. **Memory-based Function Approximators.** In this method algorithm stores the database of experiences. Every experience record contains state, action and estimated Q-value. For every state and action the expected Q-value is approximated as a weighted average of Q-values in proximity. Further details might vary. See [SS98] for the details about the method.
5. **Q-learning and neural networks.** There is a large variety of methods that combine neural network approaches and Q-learning to achieve the generalization to continuous state space and, sometimes, to continuous action space. For example, in [To97] author suggested a method for integration of Q-learning and Kohonen's self-organizing maps (SOM). Another kind of SOM and Q-learning integration was suggested in [S02]. In [GW99] authors provided an overview of neural field Q-learning and Q-radial basis approach. See [Ha99][Ha09] for more details on neural network theory.

To summarize, there is a number of methods that can generalize Q-learning approach for continuous set of states and actions, but there are much less methods that can provide efficient action choice for continuous action space.

7.2 Actor-Critic Approach in Continuous Space

Actor-critic method is reinforcement learning method that separates explicit representation of the policy and the representation of the value function. Actor is the policy

representation structure – it is used to select actions. The estimated value function is usually referred to as the critic, because it evaluates ("criticizes") the actions provided by the actor. Detailed description of the approach can be found in [SB98].

In more details, on every step critic estimates the difference between expected state value and obtained one (formula (4)).

$$\delta(t) = r_{t+1} + \gamma * V(s_{t+1}) - V(s_t) \quad (4)$$

In formula (4) $\delta(t)$ is error estimation, r_{t+1} is obtained immediate reward, γ is discount factor and V(s) is estimated state-value function. Critic also updates the state value function in a following manner (formula (5)). In formula (5) α stands for learning rate.

$$V(s_t) = V(s_t) + \alpha * \delta(t) \quad (5)$$

Actor, in turn, uses the information provided by critic to improve the policy (expression (6)). In formula (6) β is a step-size parameter and p(s,a) is an intermediate function that is used to calculate exact policy.

$$p(s_t,a_t) <\!\!- p(s_t,a_t) + \beta * \delta(t) \quad (6)$$

The policy is calculated according to formula (7).

$$\pi_t(s,a) = \exp(p(s,a)) / \Sigma_b \exp(p(s,b)) \quad (7)$$

In formula (7) $\pi_t(s,a)$ is a probability of choosing action *a* in state *s*. See [SB98] for more details on actor-critic approach.

So, actor-critic model learns the policy at once and does not work with explicit Q-functions. Therefore there is no need to search for Q-function maximum to choose the actions. That makes actor-critic approach a good candidate to be generalized for continuous action spaces. Here are some examples of how it can be done.

In the work [HW07] author suggested to work with any kind of parametrized function approximators to generalize to continuous state space. Consider that the state value function is generalized by the function approximator, that is parametrized by the vector Θ^V. In that case an update can be expressed by following formula (expression (8)).

$$\Theta^V{}_{t+1}(i) = \Theta^V{}_{t+1}(i) + \alpha * \delta(t) * d(V_t(s_t))/ d(\Theta^V{}_t(i)) \quad (8)$$

In formula (8) the term *i* stands for the number of component. So the value function is adjusted along the gradient proportionally to the error.

The work [HW07] provides some overview of policy update techniques. For example, in CACLA (Continuous Actor-Critic Learning Automation) algorithm updates the policy in a following manner. Consider that the actions are chosen according to the function Ac(s), and that function was approximated by the function approximator, which in turn was parametrized by the vector Θ^{AC}. It can be formalized in expression (9).

If $\delta(t)>0$: $\Theta^{Ac}{}_{t+1}(i) = \Theta^{Ac}{}_{t+1}(i) +$

$$\alpha * (a_t - Ac_t(s_t)) * d(Ac_t(s_t))/ d(\Theta^{Ac}{}_t(i)) \quad (9)$$

So, update depend only on the sign of $\delta(t)$, not on the value itself. Probability is updated only if the chosen action lead to the improvement of value function (better expected reward). The probabilities of choosing other actions in that state scaled down accordingly. According to [HW07], positive updates will make the approximator move towards better action, negative updates will result in moving away from worse actions, and the consequences of such drift are unknown. That's why negative updates are not used and just ignored.

Some further enhancements of those approaches include using the variance to see whether the action was exceptionally good or exceptionally bad and adjust accordingly.

To summarize, for actor-critic approach some improvements were developed, that allow it to be used in pervasive computing area.

8 Conclusion and Future Work

In this work we addressed the task of proactive adaptation to predicted context and proposed reinforcement learning based architecture, which will allow pervasive computing systems to address more complicated use cases.

The features of proactive adaptation task in pervasive computing systems were identified. The architecture of proactive adaptation solution – CALCHAS system – was developed to address them.

Context spaces theory was used as particular example of integration between context awareness system and proactive adaptation solution. The reinforcement learning algorithms capable of being applied to pervasive computing area were addressed.

Our future plans are to implement the entire middleware with a comprehensive library of available reinforcement learning approaches. Also the plan is to extend reinforcement learning solutions methods to suit the core challenges pervasive computing area.

Chapter XI

Conclusion

Conclusion

1 Thesis Summary and Discussion

This thesis proposes multiple contributions to the area of context awareness and situation awareness. Section 1 summarizes and discusses in details the contribution of the thesis. This summarizes the research questions and the answers, proposed and evaluated in the thesis.

The research questions, which became the main focus of this thesis, can be answered as follows.

Question 1: How to derive a mapping between context information and ongoing situations?

In general, the mappings between context and situations can be derived in three ways: it can be defined by expert, learned from labeled data or learned from unlabeled data. Chapter I proposes a suitable classification of mapping techniques and provides the necessary background information. Different ways of deriving the mapping have their distinct challenges, and the thesis proposes multiple ways to solve those challenges. The thesis proposes solutions to those challenges in chapters III-V. Chapter III addresses the challenges of applying expert knowledge to derive the mapping between context and situations. Chapter IV and V address relatively unexplored area of deriving the mapping from unlabeled data.

The major challenges of defining the situations manually are:

- Clarity. Human expert should be able to define the mapping between situation and context information.
- Flexibility. Modeling technique should be able to represent a wide range of real-life situations.
- Reasoning complexity. Pervasive system should be able to detect those sitautions in real time.

Chapter III [BZ11b] proposes and evaluates orthotope-based situation spaces - a set of clear and flexible situation models, which can be defined by human expert and later automatically tested. This thesis proposes orthotope-based situation representations, analyzes their flexibility aspects and evaluates reasoning complexity.

The main challenge of learning the situations from unlabeled data is detecting a situation – pervasive system should understand that some situation of interest occurs. Another major challenge is labeling a situation – pervasive system needs to give meaningful name to newly detected situation in order to present it to the user or to any program that uses situation awareness results.

Chapters IV [BZ12b] and V [KB12] address challenges of learning the situations from unlabeled data. Chapter IV uses a location awareness scenario as an example of the approach. The chapter proposes and justifies density-based clustering to infer locations of

interest out of fused location information. Also chapter IV proposes a mechanism to label the newly acquired situation in a meaningful manner. The proposed labeling technique is based on analysis of locations (retrieving nearby places of interest from Google Places API [GP12]) and analysis of time that the user spends at the place.

Chapter V [KB12] aims to learn locations and activities out of unlabeled data. Chapter IV and chapter V both use density-based clustering to infer the locations of interest. However, the other aspects of the approach are not similar. Activities in the article [KB12] are inferred by analyzing consecutive time spent at a place. In contrast with chapter IV, chapter V allows manual labeling of the learned situations. Sometimes manual labeling breaks non-obtrusiveness principle of pervasive computing, but in the lifelogging scenarios like the one addressed in chapter V manual labeling is a viable solution. Among other challenges, chapter V addresses the challenge of describing newly acquired situation to the user. In order to manually label the newly learned situation, user needs to understand precisely what situation was learned. Chapter V solves this problem by meaningfully specifying a location and by using the pictures of an activity.

The proposed situation awareness solutions are implemented as part of ECSTRA (Enhanced Context Spaces Theory-based Reasoning Architecture). ECSTRA is a general purpose context awareness and situation awareness framework, which was developed as part of PhD project. The architecture and basic functionality of ECSTRA framework is described in details in chapter II.

Once the mapping between context and situations is defined, the subsequent challenge is to prove that the derived mapping is correct. If any error is introduced into situation definition at the design time, it can lead to inadequate situation awareness at the runtime and, as a result, to improper adaptation actions. The challenge of ensuring correctness constitutes the research question 2.

Question 2: How to prove, that the derived mapping between context features and situation is correct?

The thesis answers this research question by proposing, proving and evaluating novel technique of detecting modeling errors – verification of situation models. Verification of situations was inspired by the verification of protocols [CG99], and like verification of protocols it is based on formally specifying expected property and either rigorously proving that definitions comply with the property, or algorithmically generating counterexamples – particular features of context that will lead to situation awareness inconsistency.

The detailed answer to the research question 2 is provided in chapters VI and VII. Chapter VI proposed the principle of verification, introduced basic concepts and proposed, implemented and evaluated verification algorithm for the context spaces approach [PL08a][PL08b]. Also chapter VI provides the background for chapter VII. Chapter VII [BZ12c] proposes, proves and evaluates a novel verification techinue for fuzzy situation models. Chapter VI and chapter VII propose similar first step of solving the problem – represent the property under verification as an emptiness check of situation algebra expression. However, the mentioned emptiness check requires different algorithms for the situation spaces of context spaces approach and for fuzzy situations. Practically, algorithms proposed in chapter VI and chapter VII serve somewhat similar purpose, but have nothing in common due to the differences in the task definition.

Verification of situation models is implemented as part of ECSTRA framework. ECSTRA-based implementation was used for practical evaluation of the proposed solution.

After the situation models are defined and verified, the subsequent question is how to predict future sitautions and how to properly act according to prediction results. This challenge constitutes the research question 3.

Question 3: How to predict future situation and how to act according to prediction results?

This thesis applies and implements a number of machine learning based solutions to situation prediction problem. Chapter IX [BZ09] proposes a number of methods to combine spatial representation of context (and related situation awareness techniques) with situation prediction methods. Situation prediction techniques are applied in combination with situation awareness using context spaces approach, which became a framework for answering the research questions 1 and 2.

Additionally this thesis identifies and addresses a challenge of properly acting according to predicted results. The algorithms for predicting the situations and for choosing the actions according to prediction results are usually applied sequentially, which is not applicable if adaptation actions influence prediction results. In chapter X and XI this thesis proposes general solution to the problem – combining context prediction and proactive adaptation into a single reinforcement learning task. Also this thesis introduces a set of possible reinforcement learning techniques that can be applied. In addition this thesis proposes CALCHAS (Context Aware Long-term aCt aHead Adaptation System) context prediction and proactive adaptation framework. CALCHAS is now integrated as an extension over ECSTRA.

To summarize, this thesis has three major contributions:

- This thesis proposes, proves and evaluates methods for modeling situations, unsupervised learning of situations and auto-labeling the learned situations.
- This thesis proposes, proves and evaluates methods for situation verification – a novel approach to ensure correctness of situation models.
- This thesis proposes and evaluates novel techniques for situation prediction and proactive adaptation. The issue of properly acting according to predicted context is investigated.

Next sections discuss research progress between the licentiate thesis and this dissertation, and suggest plausible directions for future research.

2 Research Progress

Significant research work was performed between the completion of licentiate thesis and the completion of PhD thesis. Chapter II, chapter VIII, chapter IX and chapter X did participate in licentiate thesis, and the modification between licentiate and PhD thesis was minor. Licentiate thesis did contain the parts corresponding to chapter III and chapter VI of this thesis, but the contents underwent major revision when transitioning from licentiate to PhD thesis. Chapter I, chapter IV, chapter V and chapter VII do not have any analogues in licentiate thesis. As a result of additional research efforts and publications, the size of PhD thesis is almost double the size of licentiate thesis.

Licentiate thesis was focused on the problems of context prediction and proactive adaptation in pervasive computing. Situation awareness enhancements were viewed as tools to enhance context prediction capabilities. However, as the research progressed and evolved, it became clear that context prediction needs a very solid backbone of proper context awareness and situation awareness methods. Situation awareness turned out to be of particular importance for many practical applications, and, moreover, many context prediction systems considered viewed context prediction as situation prediction task. As a

result, situation awareness became new main focus of the research, and main contributions of the thesis are in the area of situation awareness.

The licentiate thesis [Bo11] proposed the following future work directions, which became the basis of this thesis.

- Enhanced situation models. This direction future work was addressed in chapters IV and V of the thesis. Moreover, that direction was addressed by major improvenents in chapter III.

- Automated situation specification. Chapter IV and chapter V of the thesis propose novel techniques for automated specification of situation, thus addressing that aspect of previously proposed future work.

- Improvement of context prediction and proactive adaptation techniques. Although not mentioned directly, this aspect was improved by improving situation awareness capabilities. Situation awareness is a backbone for situation prediction, and comparing to licentiate thesis [Bo11] situation awareness was improved in many aspects. Those include situation modelling, situation inference and situation verification.

- Quality of context evaluation. This challenge remains a possible direction for future research.

In addition to following the future work directions proposed in [Bo11], this thesis contains the following achievements:

- Situation verification technique was developed further. Novel verification techniques were introduced for fuzzy situation models.

- ECSTRA framework was under constant development and improvement throughout the whole course of PhD studies. As one of the results, ECSTRA was integrated into smart home solutions developed by INRIA [DP11]. In particular, ECSTRA was appreciated for ease of use, the support of distributed and heterogeneous architectures and for flexible situation awareness mechanisms. Appendix contains a statement of accomplishment and appreciation from INRIA.

This section concludes the discussion and summary of thesis results. Next section proposes plausible future work directions for subsequent research.

3 Possible Future Work Directions[14]

Possible directions of the future work are proposed throughout the thesis. Most important ones are summarized in this section. Future work can include the following directions of research.

Quality of context evaluation. Inconsistent context information can mislead situation awareness, context prediction and proactive adaptation. Testing and formal verification of context model at the design time should be complemented with the context quality evaluation at the runtime.

Automated situation specification. This challenge is partially addressed in the thesis, but there are multiple directions for further improvement. Automated situation specification problem encompasses two separate problems: automated situation specification at the design time, using the available sources of knowledge, and automated specification at the run-time by detection and automated labeling possible situations of interest. Automated

[14] This section is partially based on the licentiate thesis [Bo11].

specification methods can improve the reliability and dependability of situation awareness mechanisms.

Improvement of situation prediction and proactive adaptation methods. In this thesis much work has been done in the area of situation prediction and proactive adaptation. However, still there is a room for improvement regarding, for example, introduction of new context prediction and proactive adaptation algorithms or addressing distributed and resource-constrained nature of pervasive conputing systems.

Introduction of new verification methods. This thesis introduces the area of formal verification of situation models, defines ground principles and proposes the algorithms for multiple kinds of situations. However, as follows from chapters VI and VII, sometimes verification algroithms significantly depend on the chosen situation modeling method. Reducing the dependency can be a direction of further research. In particular, future work might include introduction of model-independent verification approaches, as well as developing verification approaches for particular classes of situation models.

The research directions discussed above can constitute a core of new subsequent research projects.

Acronyms

ACAI – Agent Context-Aware Infrastructure [KK05]

ACES – Ambient Computing and Embedded Systems project

ACHE – Adaptive Control for Home Environment [MD95]

BN – Bayesian Network

CALCHAS – Context Awareness Long-term aCt aHead Adaptation System (see chapter X)

CDMS – Context Data Management System [XP08]

ContReMAR – Context Reasoning for Mobile Activity Recognition (see chapter IV)

CSIRO – Commonwealth Scientific and Industrial Research Organization

CST – Context Spaces Theory

DBN – Dynamic Bayesian Network

DBSCAN – Density-Based Spatial Clustering of Applications with Noise [EK96].

ECORA – Extensible Context Oriented Reasoning Architecture (see [Pa06][PL08b])

ECSTRA –Enahnced Context Spaces Theory Based Reasoning Architecture

FSI – Fuzzy Situation Inference

GLPK – GNU Linear Programming Toolkit (see [Ma12]).

GPS – Global Positioning System

GSM – Global System for Mobile communications

HMM – Hidden Markov Model

INRIA – Institut National de Recherche en Informatique et en Automatique (National Institute for Research in Computer Science and Control)

JADE – Java Agent Development framework (see [BC11])

LAN – Local Area Network

LTU – Luleå Tekniska Universitet (Luleå University of Technology)

MADIP – Mobile multi-Agent based Distributed Information Platform (see [SW11])

MDC – Mobile Data Challenge

MDP – Markov Decision Process.

OPTICS – Ordering Points To Identify the Clustering Structure (see [AB99])

PARC – Xerox Palo Alto Research Center

PDA – Personal Digital Assistant

POMDP – see Partially Observable Markov Decision Process.

SOUPA – Standard Ontology for Ubiquitous and Pervasive Applications (see [CH05]).

SVM – Support Vector Machines

WLAN – Wireless LAN

XML – eXtended Markup Language

Glossary

Activity recognition –recognizing common human activities in real life settings[KH10b].

Application Space – multidimensional space, where context attributes act as dimensions. See [Pa06], [PL08] and chapters I and VI for more details.

Context – "any information that can be used to characterize situation of an entity"[DA00]

Context Attribute – context feature that can be characterized by numeric or non-numeric value. For example, in smart home environment information like air temperature, energy consumption and light level can be taken as context attributes.

Context Awareness – ability "to provide relevant information and/or services to the user, where relevancy depends on the user's task."[DA00]

Context Data – used interchangeably with the term "context" in this thesis.

Context Information –used interchangeably with the term "context" in this thesis.

Context Prediction – forecasting future context features.

Context Space – see Application Space.

Context Spaces Approach – see Context Spaces Theory.

Context Spaces Theory – a context awareness approach, which is based on representing context as multidimensional space and utilizing spatial metaphors. See [Pa06], [PL08] and chapters I and VI for more details.

Context state – a cotext spaces theory term, which means the set of all relevant context attributes at a certain time. A context state can be viewed as a point in an application space. See chapter IV for more details.

ContReMAR – Context Reasoning for Mobile Activity Recognition, ECSTRA-based location awareness and situation awareness solution. contReMAR applcaition learns the situations out of unlabeled data. See chapter IV for more details.

Dense Orthotope-based Situation Space – a modification of orthotope-based situation space. See chapter III for more details.

ECORA – Extensible Context Oriented Reasoning Architecture. Context awareness framework, based on context spaces approach. Although both ECSTRA and ECORA are based on context spaces approach, there are no reuses between those two context awareness frameworks. See [PL08][PL04a][PL04b][Pa06] for more details

ECSTRA – Enhanced Context Spaces Theory Based Reasoning Architecture. ECSTRA is a general purpose context awareness and situation awareness engine, developed as part of this doctoral research. See chapter II for more details.

Fuzzy situation –

1) In a narrow sense, FSI-based situation model.
2) In a broader sesnse, any situation model, which is based on fuzzy logic.

Lifelogging – digitally recording aspects and personal experiences of someone's life [BL07].

Mixed context attribute – context attribute, which can take both numeric and non-numeric values.

Orthotope-based Situation Space – an extension of situation space concept, proposed in this thesis. See chapters III and VI for more details. Orthotope-based situation spaces allow representing broader class of real life situations and play an important role in verification of situation models.

Proactive adaptation – adaptation actions in response to predicted context (as opposed to reacting on current context only).

Situation – "*external semantic interpretation* of sensor data" [YD12], where the *interpretation* means "situation assigns meaning to sensor data" [YD12] and *external* means "from the perspective of applications, rather than from sensors"[YD12]. The concept of a situation generalizes the context data and elicits the most important information from it. Properly designed situation awareness extracts the most relevant information from the context data and provides it in a clear manner. Multiple aspects of situation awareness are the focus of this thesis.

Situation Awareness – the process of inferring situations out of context data.

Situation Space – formal model of a situation, which is used in context sapces theory.

Sparse Orthotope-based Situation Space – a modification of orthotope-based situation space. See chapter III for more details.

References

[AN04] Abbeel, P. and Ng, A.Y. Apprenticeship learning via inverse reinforcement learning. in *21st International Conference on Machine Learning*, 2004, pp 1-8.

[AG10] Acampora, G., Gaeta, M., Loia, V. and Vasilakos, A. V. Interoperable and adaptive fuzzy services for ambient intelligence applications. *ACM Trans. Auton. Adapt. Syst.*, vol. 5, no. 2, pp. 8:1–8:26, May 2010.

[AS94] Agrawal, R. and Srikant, R. Fast algorithms for mining association rules. in *20th Int. Conf. Very Large Data Bases*, VLDB, 1994, p. 487-499.

[AD04] Al-Bin-Ali, F. and Davies, N. Applying Logistic Regression for Activity Recognition. in *UbiComp: 6th International Conference on Ubiquitous Computing*, Poster session, 2004.

[AM06] Al-Masri, E. and Mahmoud. Q.H. A context-aware mobile service discovery and selection mechanism using artificial neural networks. in *8th international conference on electronic commerce: The new e-commerce: innovations for conquering current barriers, obstacles and limitations to conducting successful business on the internet*, 2006, p. 594-598.

[AZ98] Albrecht, D.W., Zukerman, I. and Nicholson, A.E. Bayesian models for keyhole plan recognition in an adventure game. *User modeling and user-adapted interaction*, Vol. 8, 1998, pp. 5–47.

[AZ99] Albrecht, D.W., Zukerman, I. and Nicholson, A.E. Pre-sending documents on the WWW: A comparative study. in *IJCAI99 – Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.

[AB07] Alvares, L. O., Bogorny, V., Kuijpers, B., de Macedo, J. A. F., Moelans, B. and Vaisman, A. A model for enriching trajectories with semantic geographical information. in *GIS'07: Proceedings of the 15th Annual ACM International Symposium on Advances in Geographic Information Systems*, New York, NY, USA, 2007. ACM Press.

[AM05] Anagnostopoulos, C., Mpougiouris, P. and Hadjiefthymiades, S. Prediction intelligence in context-aware applications. in *MDM '05: Proceedings of the 6th international conference on mobile data management*. New York, NY, USA: ACM Press, 2005, pp 137–141.

[AN06] Anagnostopoulos, C., Ntarladimas, Y. and Hadjiefthymiades, S. Situational computing: an innovative architecture with imprecise reasoning, *Journal of System and Software* 80 (12) (1993–2014).

[AA11] Andrienko, G., Andrienko, N., Hurter, C., Rinzivillo, S. and Wrobel, S. From Movement Tracks through Events to Places: Extracting and Characterizing Significant

Places from Mobility Data. in *IEEE Visual Analytics Science and Technology (VAST 2011) Proceedings*, IEEE Computer Society Press, pp.161-170.

[AB99] Ankerst, M., Breunig, M. M., Kriegel, H.-P. and Sander, J. OPTICS: ordering points to identify the clustering structure. in *Proceedings of the 1999 ACM SIGMOD international conference on Management of data (SIGMOD '99).* ACM, New York, NY, USA, 49-60.

[AC09] Arapinis, M., Calder, M., Dennis, L., Fisher, M., Gray, P., Konur, S., Miller, A., Ritter, E., Ryan, M., Schewe, S., Unsworth, C. and Yasmin, R.. Towards the Verification of Pervasive Systems. *Electronic Communication of the European Association of Software Science and Technology* 22, 2009.

[AS02] Ashbrook, D. and Starner, S. Learning Significant Locations and Predicting User Movement with GPS. in *Proceedings of the 6th IEEE International Symposium on Wearable Computers*, 2002, p.101.

[AL08] Augusto, J.C., Liu, J., McCullagh, P., Wang, H. and Yang, J.-B. Management of uncertainty and spatio-temporal aspects for monitoring and diagnosis in a smart home. *International Journal of Computational Intelligence Systems* 1 (4) ,2008, pp 361–378.

[BC93] Baird, L. C. and Klopf, A. H. Reinforcement learning with high-dimensional, continuous actions. *Wright-Patterson Air Force Base Ohio: Wright Laboratory, Tech. Rep. WL-TR-93-1147*, 1993. URL= http://www.leemon.com/papers/1993bk3.pdf

[BI04] Bao, L. and Intille, S.S. Activity recognition from user-annotated acceleration data. in: *Pervasive'04: Proceedings of the Second International Conference on Pervasive Computing*, Vienna, Austria, April 2004, pp. 1–17.

[BP66] Baum, L.E. and T. Petrie. Statistical inference for probabilistic functions of finite state Markov chains. *The Annals of Mathematical Statistics*, 1966, pp. 1554–1563.

[BC11] Bellifemine, F., Caire, G., Trucco, T. and Rimassa, G. JADE Programmer's Guide, URL=http://jade.tilab.com/doc/programmersguide.pdf, last accessed October, 30, 2012.

[Be96] Bestavros, A. Speculative data dissemination and service to reduce server load, network traffic and service time in distributed information systems. in *Proceedings of the 1996 International Conference on Data Engineering*, 1996.

[BB10] Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A. and Riboni, D. A survey of context modelling and reasoning techniques, *Pervasive and Mobile Computing* 6 (2), 2010, pp 161–180.

[BD99] Bhattacharya, A. and Das, S.K. Lezi update: An information-theoretic approach to track mobile users in PCS networks. *Mobile Computing and Networking*, 1999, pp 1–12.

[BV04] Boyd, S. P. and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, 2004.

[Bo10] Boytsov, A. Proactive Adaptation in Pervasive Computing Systems, in *ICPS '10: Proceedings of the 7th international conference on Pervasive services*, Berlin, Germany: ACM, 2010.

[Bo11] Boytsov, A. Context Reasoning, Context Prediction and Proactive Adaptation in Pervasive Computing Systems. *Licentiate thesis. Department of Computer Science, Electrical and Space Engineering*, Luleå University of Technology, 2011. URL=http://pure.ltu.se/portal/files/32946690/Andrey_Boytsov.Komplett.pdf, last accessed October, 30, 2012.

[BZ09] Boytsov, A., Zaslavsky, A. and Synnes, K. Extending Context Spaces Theory by Predicting Run-Time Context, in *Proceedings of the 9th International Conference on Smart Spaces and Next Generation Wired/Wireless Networking and Second Conference on Smart Spaces*. St. Petersburg, Russia: Springer-Verlag, 2009, pp. 8-21.

[BZ10a] Boytsov, A. and Zaslavsky, A. Context prediction in pervasive computing systems: achievements and challenges. in Burstein, F., Brézillon, P. and Zaslavsky, A. eds. *Supporting real time decision-making: the role of context in decision support on the move*. Springer p. 35-64. 30 p. (Annals of Information Systems; 13), 2010.

[BZ10b] Boytsov, A. and Zaslavsky, A. Extending Context Spaces Theory by Proactive Adaptation. in Balandin, S., Dunaytsev, R. and Koucheryavy, Y. eds. *Proceedings of the 10th international conference and 3rd international conference on Smart spaces and next generation wired/wireless networking (NEW2AN'10/ruSMART'10)*, Springer Berlin / Heidelberg, 2010, pp. 1-12.

[BZ11a] Boytsov, A. and Zaslavsky, A. ECSTRA: distributed context reasoning framework for pervasive computing systems. in Balandin, S., Koucheryavy, Y. and Hu H. eds. *Proceedings of the 11th international conference and 4th international con ference on Smart spaces and next generation wired/wireless networking (NEW2AN'11/ruSMART'11)*, Springer-Verlag, Berlin, Heidelberg, 1-13.

[BZ11b] Boytsov, A. and Zaslavsky, A. From Sensory Data to Situation Awareness: Enhanced Context Spaces Theory Approach, in *Proceedings of IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC)*, 2011 , pp.207-214, 12-14 Dec. 2011. doi: 10.1109/DASC.2011.55.

[BZ11c] Boytsov, A. and Zaslavsky, A. Formal Verification of the Context Model - Enhanced Context Spaces Theory Approach. Scientific report, 2011, 41 p. URL=http://pure.ltu.se/portal/files/32810947/BoytsovZaslavsky_Verification_TechReport.pdf, last accessed October, 30, 2012.

[BZ12a] Boytsov, A., Zaslavsky, A. and Abdallah, Z. Where Have You Been? Using Location Clustering and Context Awareness to Understand Places of Interest. in Andreev, S., Balandin, S. and Koucheryavy, Y. eds. *Internet of Things, Smart Spaces, and Next Generation Networking*, vol. 7469, Springer Berlin / Heidelberg, 2012, pp. 51–62.

[BZ12b] Boytsov, A. and Zaslavsky, A. Formal verification of context and situation models in pervasive computing. *Pervasive and Mobile Computing*, Volume 9, Issue 1, February 2013, Pages 98-117, ISSN 1574-1192, 10.1016/j.pmcj.2012.03.001. URL=http://www.sciencedirect.com/science/article/pii/S1574119212000417, last accessed May, 08, 2013.

[BZ12c] Boytsov, A. and Zaslavsky, A. Correctness Analysis and Verification of Fuzzy Situations in Situation Aware Pervasive Computing Systems. Scientific report, 2013. URL=http://pure.ltu.se/portal/files/42973133/BoytsovZaslavsky_FuzzyVerifReport.pdf, last accessed May, 08, 2013.

[BU98] Burges, C. J. C. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, v.2 n.2, p.121-167, June 1998. DOI = 10.1023/A:1009715923555

[BD08] Byrne, D., Doherty, A.R., Snoek C.G.M., Jones, G. J. F., and Smeaton, A.F. Validating the Detection of Everyday Concepts in Visual Lifelogs. *SAMT 2008 – 3rd International Conference on Semantic and Digital Media Technologies*, Koblenz, Germany, 3-5 D, 2008.

[BL07] Byrne, D., Lavelle, B., Doherty, A. R. Jones, G. J. F. and Smeaton, A. F. Using Bluetooth & GPS Metadata to Measure Event Similarity in SenseCam Images. *Centre for Digital Video Processing (CDVP) & Adaptive Information Cluster (AIC)*, Dublin City University, Dublin 9, Ireland, 2007.

[CG09] Calder, M. Gray, P. and Unsworth, C. Tightly coupled verification of pervasive systems, in *Proceedings of the Third International Workshop on Formal Methods for Interactive Systems (FMIS 2009)*, 2009.

[CX05] Cao, J., Xing, N., Chan, A., Feng, Y. and Jin, B. Service Adaptation Using Fuzzy Theory in Context-aware Mobile Computing Middleware. in *Proceedings of the 11th IEEE Conference on Embedded and Real-time Computing Systems and Applications (RTCSA 2005)*, 2005.

[CM05] Capper, O., Moulines, E. and Rydern, T. *Inference in Hidden Markov Models*, Springer, 2005.

[CG00] Cardelli, L. and Gordon, A.D. Mobile ambients. *Theoretical Computer Science*, vol. 240, Jun. 2000, pp. 177-213.

[Ch97] Chalfen R. Family photography: One album is worth a 1000 lies. in Neuwman, D. M. ed. *Sociology: Exploring the architecture of everyday life*, CA.: Pine Forge Press, 1997, pp. 269-278.

[CF05] Chen, H., Finin, T. and Joshi, A. The SOUPA Ontology for Pervasive Computing. in Tamma, V., Cranefield, S. and Finin, T. eds. *Ontologies for Agents: Theory and Experiences*, Springer, 2005.

[CG99] Clarke, E. M., Grumberg, Orna and Peled, D. A. *Model checking*. Cambridge, Mass., London : MIT, 1999.

[CD05] Cook, D.J. and Das, S.K. *Smart environments: technologies, protocols, and applications*, Wiley-Interscience, 2005.

[CD07] Cook, D.J. and Das, S.K. How smart are our environments? An updated look at the state of the art. in *Pervasive and Mobile Computing*, vol. 3, 2007, pp. 53-73.

[CA09] Cook, D.J., Augusto, J.C. and Jakkula, V.R. Ambient intelligence: Technologies, applications, and opportunities. in *Pervasive and Mobile Computing*, vol. 5, Aug. 2009, pp. 277-298.

[CL09] Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. *Introduction to Algorithms (3rd ed.)*. MIT Press, 2009. ISBN 0-262-03384-4.

[CP10] Coronato, A. and Pietro, G.D. Formal specification of wireless and pervasive healthcare applications. *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 10, Aug. 2010, pp. 12:1–12:18.

[CP11] Coronato, A. and Pietro, G.D. Formal Specification and Verification of Ubiquitous and Pervasive Systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 6, Feb. 2011, pp. 9:1–9:6.

[Co73] Coxeter, H. S. M. *Regular Polytopes*, 3rd ed. New York: Dover, 1973.

[DC02] Das, S.K., Cook, D.J., Bhattacharya, A., Heierman, E.O. and Lin, T.Y. The role of prediction algorithms in the MavHome smart home architecture. *IEEE Wireless Communications*, Vol. 9, 2002, pp. 77–84.

[DH97] Davison, B. D. and Hirsh, H. Toward an adaptive command line interface. in *Proceedings of the Seventh International Conference on Human-Computer Interaction*, San Francisco, CA: Elsevier Science Publishers, 1997.

[DH98] Davison, B.D. and Hirsh, H. Probabilistic online action prediction. in *Proceedings of the AAAI Spring Symposium on Intelligent Environments*, 1998, pp. 148–154.

[DZ08] Delir Haghighi, P., Zaslavsky, A., Krishnaswamy, S., and Gaber, M.M. Reasoning About Context in Uncertain Pervasive Computing Environments. in *Proceedings of the European Conference on Context and Sensing (EuroCSS08)*, Switzerland, October, Springer Verlag, 2008.

[DL77] Dempster, A. P., Laird, N. M. and Rubin, D. B. Maximum Likelihood from Incomplete Data via the EM algorithm. *Journal of the Royal Statistical Society*, vol. 39, no. 1, 1977, pp. 1–38.

[DA00] Dey, A.K. and Abowd, G.D. Towards a better understanding of context and context-awareness. in *CHI 2000 workshop on the what, who, where, when, and how of context-awareness*, 2000, pp. 304–307.

[DS07] Dimakis, N., Soldatos, J. and Polymenakos, L. Ontology-based Management of Pervasive Systems. in *Proceeding of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, IOS Press, 2007, pp. 106-113.

[DP07] Dimitrov, T., Pauli, J. and Naroska, E. A probabilistic reasoning framework for smart homes. in *Proceedings of the 5th international workshop on Middleware for pervasive and ad-hoc computing: held at the ACM/IFIP/USENIX 8th International Middleware Conference*, New York, NY, USA, 2007, pp. 1–6.

[Do09] Doherty, A. R. Providing effective memory retrieval cues through automatic structuring and augmentation of a lifelog of images. *PhD thesis*, Dublin City University, 2009.

[DC08] Doherty, A.R, Ó Conaire, C., Blighe, M., Smeaton, A.F. and O'Connor, N. Combining Image Descriptors to Effectively Retrieve Events from Visual Lifelogs. In

MIR 2008 – ACM International Conference on Multimedia Information Retrieval, Vancouver, Canada, 2008, pp30-31.

[DP11] Dominici, M., Pietropaoli, B. and Weis. F. Towards a feasibility-driven uncertainty-aware layered architecture for recognizing complex domestic activity. In *Proceedings of the 2011 international workshop on Situation activity & goal awareness(SAGAware '11)*. ACM, New York, NY, USA, pp 89-94, 2011. DOI=10.1145/2030045.2030064, URL=http://doi.acm.org/10.1145/2030045.2030064, last accessed October, 30, 2012.

[Do07] Dougherty, C. *Introduction to Econometrics*. Third edition, Oxford University Press, 2007.

[Ea11] Eaton, J. W. *GNU Octave Manual*, Network Theory Ltd., 2002.

[ES07] Ejigu, D., Scuturici, M. and Brunie, L. An Ontology-Based Approach to Context Modeling and Reasoning in Pervasive Computing. in *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, IEEE Computer Society, 2007, pp. 14-19.

[En01] H. B. Enderton. A Mathematical Introduction to Logic. Elsevier Science, 2001.

[EK96] Ester, M., Kriegel, H.-P., Sander, J. and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. in *Proc KDD. AAAI Press*, Volume: 96, 1996, Pages: 226-231. ISBN: 1577350049.

[FT07] Favela, J., Tentori, M., Castro, L. A., Gonzalez, V. M., Moran, E. B. and Martínez-García, A. I. Activity recognition for context-aware hospital applications: issues and opportunities for the deployment of pervasive networks. *Mob. Netw. Appl.*, vol. 12, no. 2–3, Mar. 2007, pp. 155–171.

[FM92] Feder, M., Merhav, N. and Gutman, M. Universal prediction of individual sequences. *IEEE Transactions on Information Theory*, Vol. 38, 1992, pp. 1258–1270.

[FL07] Feki, M., Lee, S., Bien, Z. and Mokhtari, M. Context Aware Life Pattern Prediction Using Fuzzy-State Q-Learning. in *Pervasive Computing for Quality of Life Enhancement*, pp. 188-195, 2007.

[FK02] Frohlich, D., Kuchinsky, A., Pering, C., Don, A. and Ariss, S.. Requirements for photoware. *In Proc. of CSCW'02*, ACM Press, New Orleans, Louisiana, USA, 2002.

[Fr95] Fritzke, B. A growing neural gas network learns topologies. *Advances in Neural Information Processing Systems* 7, 1995, pp. 625-632.

[GP09] Gammage, B., Plummer, D. C., Thompson, E., Fiering, L., LeHong, H., Karamouzis, F., Da Rold, C., Collins, K., Clark, W., Jones, N., Smulders, C., Escherich, M., Reynolds, M. and Basso, M. Gartner's Top Predictions for IT Organizations and Users, 2010 and Beyond: A New Balance, 2009. URL = http://www.businessanalisten.nl/RT/0Portal/Event/doc/Gar_Top_pred_2010.pdf, last accessed October, 30, 2012.

[GW99] Gaskett, C., Wettergreen, D., Zelinsky, A. and Zelinsky, E. Q-Learning in Continuous State and Action Spaces. in *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence*, 1999, pp. 417-428.

[GV06] Gellert, A. and Vintan, L. Person Movement Prediction Using Hidden Markov Models. *Studies in Informatics and Control*, Vol. 15, 2006, p. 17.

[GW04] Gemmell, J., Williams, L., Wood, K., Bell, G. and Lueder, R. Passive Capture and Ensuing Issues for a Personal Lifetime Store. in *Proceedings of The First ACM Workshop on Continuous Archival and Retrieval of Personal Experiences (CARPE '04)*, New York, NY, USA, 2004, pp. 48-55.

[GR04] Giarratano, J.C. and Riley, G. *Expert systems: principles and programming*, Course Technology, USA, 2004.

[GC03] Gopalratnam, K. and Cook, D.J. Active LeZi: An incremental parsing algorithm for sequential prediction. in *Proceedings of the Florida Artificial Intelligence Research Symposium*, 2003, pp. 38–42.

[GH12] Gordon, D., Hanne, J.-H., Berchtold, M., Miyaki, T. and Beigl, M. Recognizing Group Activities Using Wearable Sensors. in Puiatti, A. and Gu, T. eds. *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, vol. 104, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 350–361.

[GG06] Gottfried, B., Guesgen, H.W., Hübner, S. Spatiotemporal reasoning for smart homes. *Designing Smart Homes* 4008, 2006, pp 16–34.

[Gr93] Gruber, T. R. A translation approach to portable ontology specifications. *Knowledge Acquisition*, vol. 5, no. 2, Jun. 1993, pp. 199–220.

[HD99] Hagan, M.T. and Demuth, H.B. Neural networks for control. in *Proceedings of the 1999 American Control Conference*, San Diego, CA, 1999, pp. 1642–1656.

[HF09] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. and Witten, I. H. The WEKA data mining software: an update, *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009.

[HM06] Hamid, R., Maddi, S., Bobick, A. and Essa, I. Unsupervised analysis of activity sequences using event-motifs, in *VSSN'06: Proceedings of the 4th ACM International Workshop on Video Surveillance and Sensor Networks*, ACM, New York, NY, USA, 2006, pp. 71–78.

[Ha01] Hamker, F. H. Life-long learning cell structures—continuously learning without catastrophic interference. *Neural Networks* 14, no. 4–5, 2001, pp 551–573.

[HL08] Han, J., Lee, J.-G., Gonzalez, H. and Li, X. Mining Massive RFID, Trajectory, and Traffic Data Sets (Tutorial). in *KDD*, 2008.

[HM93] Harris, C. J., Moore, C. G. and Brown, M. *Intelligent control: aspects of fuzzy logic and neural nets*, vol. 6. World Scientific Pub Co Inc, 1993.

[HW07] Hasselt, H. van and Wiering, M. Reinforcement Learning in Continuous Action Spaces. in *Proceedings of IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL07)*, Honolulu, HI, USA, 2007, pp. 272-279.

[Ha99] Haykin, S. Neural Networks: A Comprehensive Foundation, 2nd ed., Prentice Hall, 1999.

[Ha09] Haykin, S. *Neural Networks and Learning Machines*. 3rd edition, Pearson Education, NJ, 2009.

[Ho62] Hoare, C. A. R. Quicksort. *The Computer Journal*, vol. 5, no. 1, 1962, pp. 10–16.

[HW06] Hodges, S., Williams, L., Berry, E., Izadi, S., Srinivasan, J., Butler, A., Smyth, G., Kapur, N. and Wood, K. SenseCam: A retrospective memory aid. in *UbiComp: 8th International Conference on Ubiquitous Computing*, volume 4602 of LNCS.

[HB07] Hoey, J., Bertoldi, A. von, Poupart, P. and Mihailidis, A. Assisting persons with dementia during handwashing using a partially observable Markov decision process. in *Proceedings of the Int. Conf. on Vision Systems*, 2007, p. 66.

[Ho97] Holzmann, G. J. The model checker SPIN. *IEEE Transactions on Software Engineering*, vol. 23, no. 5, May 1997, pp. 279 –295.

[HN09] Hong, X., Nugent, C., Liu, W., Ma, J., McClean, S., Scotney, B. and Mulvenna, M. Uncertain information management for ADL monitoring in smart homes. *Intelligent Patient Management*, 189/2009, pp 315–332, 2009.

[HS09] Hong, J., Suh, E.H., Kim, J. and Kim, S.Y. Context-aware system for proactive personalized service based on context history. *Expert Systems with Applications*, vol. 36, 2009, pp. 7448–7457.

[HB98] Horvitz, E., Breese, J., Heckerman, D., Hovel, D. and Rommelse, K. The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, 1998, pp. 256–265.

[HL00] Hosmer, D.W. and Lemeshow, S. *Applied Logistic Regression*. John Wiley & Sons, 2000.

[IN09] Inomata, T., Naya, F., Kuwahara, N., Hattori, F. and Kogure, K. Activity recognition from interactions with objects using dynamic Bayesian network. in *Proceedings of the 3rd ACM International Workshop on Context-Awareness for Self-Managing Systems*, New York, NY, USA, 2009, pp. 39–42.

[IS09] Ishikawa, F., Suleiman, B., Yamamoto, K. and Honiden, S. Physical interaction in pervasive computing: formal modeling, analysis and verification. in *Proceedings of the 2009 International Conference on Pervasive Services*, New York, NY, USA: ACM, 2009, pp. 133–140.

[JY08] Jeung, H., Yiu, M. L., Zhou, X., Jensen, C. S. and Shen, H. T. Discovery of Convoys in Trajectory Databases. *VLDB*, pages 1068–1080, 2008.

[KL96] Kaelbling, L.P., Littman, M.L. and Moore, A.W. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, vol. 4, 1996, pp. 237–285.

[KS10] Kalnikaite, V., Sellen, A., Whittaker, S. and Kirk, D. Now Let Me See Where I Was: Understanding How Lifelogs Mediate Memory. *CHI 2010*, ACM Press, Atlanta, GA, USA.

[KG08] Kanda, T., Glas, D.F., Shiomi, M., Ishiguro, H. and Hagita, N. Who will be the customer?: a social robot that anticipates people's behavior from their trajectories. in

UbiComp'08: Proceedings of the 10th International Conference on Ubiquitous Computing, ACM, Seoul, Korea, 2008, pp. 380–389.

[KL02] Kaowthumrong, K., Lebsack, J. and Han, R. Automated selection of the active device in interactive multi-device smart spaces. in *UbiComp'02: Supporting Spontaneous Interaction in Ubiquitous Computing Settings*, 2002.

[KB03] Karbassi, A. and Barth, M. Vehicle Route Prediction and Time of Arrival Estimation Techniques for Improved Transportation System Management. in *Intelligent Vehicles Symposium*, 2003. p. 511-516.

[K84] Karmarkar, N. A new polynomial-time algorithm for linear programming. in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, 1984, pp. 302–311.

[KA08] Kasteren, T. van, Athanasios, N., Englebienne, G. and Kröse, B. J. A. Accurate activity recognition in a home setting. in *UbiComp '08: Proceedings of the 10th international conference on Ubiquitous computing*, New York, NY, USA, 2008, pp 1-9.

[KK07] Kasteren, T. van and Kröse, B. J. A. Bayesian activity recognition in residence for elders. in *IE'07: Proceedings of the third IET International Conference on Intelligent Environments*, September 2007, pp. 209–212.

[KE11] Kasteren, T. L. M., Englebienne, G. and Kröse, B. J. A. Hierarchical Activity Recognition Using Automatically Clustered Actions. Keyson, D. V., Maher, M. L., Streitz, N., Cheok, A., Augusto, J. C., Wichert, R., Englebienne, G., Aghajan, H. and Kröse, B. J. A. eds. *Ambient Intelligence*, vol. 7040, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 82–91.

[KK05] Khedr, M. and Karmouch, A. ACAI: agent-based context-aware infrastructure for spontaneous applications. *Journal of Network and Computer Applications*, vol. 28, Jan. 2005, pp. 19-44.

[KB10] Kikhia, B., Bengtsson, J. E.. Synnes, K.. ul Hussain Sani, Z. and Hallberg, J. Creating Digital Life Stories through Activity Recognition with Image Filtering. in P*roceeding of the 8th International Conference on Smart Homes and Health Telematics (ICOST)*, Seoul, South Korea, 2010, pp 203–210.

[KB12] Kikhia, B., Boytsov, A., Hallberg, J., ul Hussain Sani, Z., Jonsson, H. and Synnes, K. Structuring and Presenting Lifelogs based on Location Data. Technical report. 2012. 19p.
URL=http://pure.ltu.se/portal/files/40259696/KB12_StructuringPresentingLifelogs_TR.pdf, last accessed October, 30, 2012.

[KH10a] Kikhia, B., Hallberg, J., Bengtsson, J.E., Sävenstedt, S. and Synnes, K.. Building digital life stories for memory support. *Int. J. Computers in Healthcare*, Vol. 1, No. 2, 2010, pp.161–176.

[KH10b] Kim, E., Helal, S. and Cook, D. Human activity recognition and pattern discovery. *Pervasive Computing, IEEE*, vol. 9, no. 1, 2010, pp. 48–53.

[KK08] Kröse, B., van Kasteren, T., Gibson, C. and van den Dool, T. Care: Context awareness in residences for elderly. in *Proceedings of ISG*, 2008, pp. 101–105.

[Kl02] Kleene, Stephen Cole. Mathematical logic. Dover Publications, 2002.

[Kr07] Krumm, J. A Markov model for driver route prediction. in *Society of Automative Engineers (SAE) World Congress*, 2007.

[KW10] Kwapisz, J. R., Weiss, G. M. and Moore, S. A. Activity recognition using cell phone accelerometers. *SIGKDD Explor. Newsl.*, vol. 12, no. 2, Mar. 2011, pp. 74–82.

[LO11] Laguna, J. O., Olaya, A. G. and Borrajo, D. A dynamic sliding window approach for activity recognition. in *Proceedings of the 19th international conference on User modeling, adaption, and personalization*, Berlin, Heidelberg, 2011, pp. 219–230.

[LG12] Laurila, J. K., Gatica-Perez, D., Aad, I., Blom, J., Bornet, O., Do, T.-M.-T., Dousse, O., Eberle, J. and Miettinen, M. The Mobile Data Challenge: Big Data for Mobile Computing Research. in *Proc. Mobile Data Challenge by Nokia Workshop, in conjunction with Int. Conf. on Pervasive Computing*, Newcastle, June 2012.

[LL12] Lee, S., Lin, G., Jih, W., Huang, C.-C. and Hsu, J. Y. Energy-Aware agents for detecting nonessential appliances. in *Proceedings of the 13th international conference on Principles and Practice of Multi-Agent Systems*, Berlin, Heidelberg, 2012, pp. 475–486.

[LJ10] Li, Z., Ji, M., Lee, J.-G., Tang, L.-A., Yu, Y., Han, J. and Kays, R. MoveMine: Mining Moving Object Databases. *SIGMOD*, 2010, pp 1203–1206.

[LW08] Lin, T., Wang, C. and Lin, P.C. A neural-network-based context-aware handoff algorithm for multimedia computing. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, vol. 4, 2008, pp. 1–23.

[Lo04a] Loke, S.W. Facing Uncertainty and Consequence in Context-Aware Systems: Towards an Argumentation Approach. in *Proceedings of the Workshop on Advanced Context Modelling, Reasoning and, Management @ Ubicomp 2004*, 2004.

[Lo04b] Loke, S. W. Logic Programming for Context-Aware Pervasive Computing: Language Support, Characterizing Situations, and Integration with the Web. in *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence*, 2004. WI 2004, 2004, pp. 44 – 50.

[Ma12] Makhorin, A.. *GLPK (GNU Linear Programming Kit)*. URL=http://www.gnu.org/software/glpk/, last accessed October, 30, 2012.

[MS02] Mäntyjärvi, J. and Seppänen, T. Adapting Applications in Mobile Terminals Using Fuzzy Context Information. in Paternò, F. ed. *Human Computer Interaction with Mobile Devices*, vol. 2411, Springer Berlin / Heidelberg, 2002, pp. 383–404.

[Ma04a] Mayrhofer, R. An Architecture for Context Prediction. *PhD thesis*, Johannes Kepler University of Linz, Austria, October 2004.URL= http://www.mayrhofer.eu.org/downloads/publications/PhD-ContextPrediction-2004.pdf, last accessed October, 30, 2012.

[Ma04b] Mayrhofer, R. An Architecture for Context Prediction. in *Advances in Pervasive Computing*, Austrian Computer Society (OCG), 2004, pp. 72, 65.

[MR04] Mayrhofer, R., Radi, H. and Ferscha, A. Recognizing and predicting context by learning from user behavior. *Radiomatics: Journal of Communication Engineering, special issue on Advances in Mobile Multimedia*, 1(1), May 2004.

[MY10] McKeever, S., Ye, J., Coyle, L., Bleakley, C. and Dobson, S. Activity recognition using temporal evidence theory. *Journal of Ambient Intelligence and Smart Environments*, vol 2, issue 3, August 2010.

[Me92] Mehrotra, S. On the implementation of a primal-dual interior point method. *SIAM Journal on optimization*, vol. 2, no. 4, 1992, pp. 575–601.

[Mo98] Mozer, M.C. The neural network house: An environment that adapts to its inhabitants. in *Proceedings of the American Association for Artificial Intelligence Spring Symposium on Intelligent Environments*, 1998, pp. 110–114.

[Mo04] Mozer, M.C. Lessons from an adaptive home. in *Smart environments: technologies, protocols, and applications*, pp. 273–294. Wiley (2004)

[MD95] Mozer, M. C., Dodier, R. H., Anderson, M., Vidmar, L., Iii, R. F. C. and Miller, D. The Neural Network House: An Overview. in Niklasson, L., Boden, M. eds. *Current trends in connectionism*, Hillsdale, NJ: Erlbaum, 1995, pp. 371-380.

[MB04] Mühlenbrock, M., Brdiczka, O., Snowdon, D. and Meunier, J.-L. Learning to detect user activity and availability from a variety of sensor data. in *PERCOM'04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications*, IEEE Computer Society, Orlando, Florida, March 2004, pp. 13–22.

[NM05] Nurmi, P., Martin, M. and Flanagan, J. A. Enabling proactiviness through Context Prediction. in *Proceedings of the Workshop on Context Awareness for Proactive Systems* (CAPS, Helsinki, Finland, June, 2005), Helsinki University Press, 2005, pp 159-168.

[PL04] Padovitz, A., Loke, S. W. and Zaslavsky, A.. Towards a theory of context spaces. in *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, 2004, pp 38–42.

[Pa06] Padovitz, A. Context Management and Reasoning about Situations in Pervasive Computing. *PhD Theses*, Caulfield School of Information Technology, Monash University, Australia, 2006.

[PZ06] Padovitz, A., Zaslavsky, A. and Loke, S. W. A Unifying Model for Representing and Reasoning About Context under Uncertainty. in *11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU)*, July 2006, Paris, France.

[PL07] Padovitz, A., Loke, S.W., Zaslavsky, A. and Burg, B. Verification of Uncertain Context Based on a Theory of Context Spaces. *International Journal of Pervasive Computing and Communications*, vol. 3, issue 1, 2007, pp 30 – 56

[PL08a] Padovitz, A., Loke, S. W. and Zaslavsky, A. Multiple-agent perspectives in reasoning about situations for context-aware pervasive computing systems. *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans*, 38(4), 2008, pp 729-742.

[PL08b] Padovitz, A., Loke, S. W. and Zaslavsky, A. The ECORA framework: A hybrid architecture for context-oriented pervasive computing. *Pervasive and Mobile Computing*, Elsevier, 4, 2008, pp 182–215.

[PR07] Patel, S. N., Robertson, T., Kientz, J. A., Reynolds, M. S. and Abowd, G. D. At the flick of a switch: detecting and classifying unique electrical events on the residential power line. in *Proceedings of the 9th international conference on Ubiquitous computing*, Berlin, Heidelberg, 2007, pp. 271–288.

[PB08] Palma, A. T, Bogorny, V., Kuijpers, B. and Alvares, L.O. A Clustering-based Approach for Discovering Interesting Places in Trajectories. in: *23rd Annual Symposium on Applied Computing, (ACM-SAC'08)*, Fortaleza, Ceara, Brazil, 16-20 March 2008, pp. 863-868.

[PB03] Petzold, J., Bagci, F., Trumler, W. and Ungerer, T. Context Prediction Based on Branch Prediction Methods. *Technical report*, 2003. URL=http://www.informatik.uni-augsburg.de/lehrstuehle/sik/publikationen/reports/2003_14_pet/2003_14_pet_pdf.pdf, last accessed October, 30, 2012.

[PP05] Petzold, J., Pietzowski, A., Bagci, F., Trumler, W. and Ungerer, T. Prediction of indoor movements using bayesian networks. in *Location-and Context-Awareness (LoCA 2005)*, Oberpfaffenhofen, Germany, 2005, pp. 211-222.

[PB04] Pichler, M., Bodenhofer, U., Schwinger, W. Context-Awareness and Artificial Intelligence. *Österreichische Gesellschaft für Artificial Intelligence (ÖGAI)*, 23/1, ISSN 0254-4326, April 2004.

[Pi01] Piegat, A. *Fuzzy Modeling and Control*, 1st ed. Physica, 2001.

[Po09] Poslad, S. *Ubiquitous Computing: Smart Devices, Environment and Interactions*. John Wiley & Sons Ltd., 2009.

[Qu93] Quinlan, R. J. C4.5: Programs for Machine Learning. *Morgan Kaufmann Series in Machine Learning*, Morgan Kaufmann, January 1993.

[Ra90] Rabiner, L.R. A tutorial on hidden Markov models and selected applications in speech recognition. *Readings in Speech Recognition*, vol. 53, 1990, pp. 267–296.

[Ra07] Rakowsky, U. Fundamentals of Dempster-Shafer theory and its applications to system safety and reliability modeling. *RTA*, no. 3-4, 2007.

[RA04] Ranganathan, A., Al-Muhtadi, J. and Campbell, R. Reasoning about uncertain contexts in pervasive computing environments. *Pervasive Computing, IEEE*, vol. 3, 2004, pp. 62-70.

[RB03] Roy, A., Bhaumik, S.K., Bhattacharya, A., Basu, K., Cook, D.J. and Das, S.K. Location aware resource management in smart homes. in *Proceedings of the First*

IEEE International Conference on Pervasive Computing and Communications, 2003, pp. 481–488.

[RD03] Roy, A., Das, S.K., Bhattacharya, A., Basu, K., Cook, D. J. and Das, S. K.. Location aware resource management in smart homes. in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003. (PerCom 2003)*, 2003, pp 481–488.

[RN06] Russell, S. J. and Norvig, P. *Artificial Intelligence. A Modern Approach*, 2nd edition. Upper Saddle River, New Jersey, USA: Prentice Hall, 2006.

[RN09] Russell, S. J. and Norvig, P. *Artificial Intelligence: A Modern Approach*, 3rd edition. Prentice Hall, 2010.

[RA09] Ryoo, M. S. and Aggarwal, J. K. Semantic Representation and Recognition of Continued and Recursive Human Activities. *Int. J. Comput. Vision*, vol. 82, no. 1, April 2009, pp. 1–24.

[SS98] Santamaria, J.C., Sutton, R.S. and Ram, A. Experiments with Reinforcement Learning in Problems with Continuous State and Action Spaces. *Adaptive Behavior*, vol. 6, issue 2, 1998, pp. 163-217.

[Sh76] Shafer, G., *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, N.J., 1976.

[SH06] Sigg, S., Haseloff, S. and David, K. A Novel Approach to Context Prediction in UBICOMP Environments. in *Proceedings of the 2006 IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications*, 2006, pp. 1-5.

[SH07] Sigg, S., Haseloff, S. and David, K. Minimising the Context Prediction Error. in *Proceedings of IEEE 65th Vehicular Technology Conference VTC2007-Spring*, Dublin, Ireland, 2007, pp. 272-276.

[SL09] Siirtola, P., Laurinen, P., Haapalainen, E., Roning, J. and Kinnunen, H. Clustering-based activity classification with a wrist-worn accelerometer using basic features. in *IEEE Symposium on Computational Intelligence and Data Mining, 2009. CIDM '09*, 2009, pp. 95 –100.

[SB06] Simmons, R., Browning, B., Zhang, Y. and Sadekar, V. Learning to predict driver route and destination intent. in *IEEE Intelligent Transportation Systems Conference – ITSC'06*, 2006, pp. 127–132.

[Sm02] Smith, A.J. Applications of the self-organising map to reinforcement learning. *Neural Networks*, vol. 15, 2002. pp. 1107-1124.

[SP08] Spaccapietra, S., Parent, C., Damiani, M. L., de Macedo, J. A., Porto, F. and Vangenot, C.. A Conceptual View on Trajectories. *Data and Knowledge Engineering*, vol 65, issue 1, 2008, pp 126–146.

[SA96] Srikant, R. and Agrawal, R. Mining sequential patterns: Generalizations and performance improvements. *Advances in Database Technology – EDBT'96*, 1996, pp. 1–17.

[St09] Staab, S. *Handbook on Ontologies*. Springer-Verlag, 2009.

[SW11] Su, C. and Wu, C. JADE implemented mobile multi-agent based, distributed information platform for pervasive health care monitoring. *Applied Soft Computing*, vol 11, Jan. 2011, pp. 315-325.

[SB98] Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. The MIT press, Cambridge MA, A Bradford Book, 1998.

[TI04] Tapia, E.M., Intille, S.S. and Larson, K. Activity recognition in the home using simple and ubiquitous sensors. in: *Pervasive'04: Proceedings of the international conference on Pervasive Computing*, 2004, pp. 158–175.

[To97] Touzet, C. Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems*, vol. 22, 1997, pp. 251-281.

[TL03] Toyama, K., Logan, R., Roseway, A. and Anandan, P. Geographic Location Tags on Digital Images. in *Proceedings of the eleventh ACM international conference on Multimedia*, Berkeley, California, November 2003, 1-58113-722-2.

[TW08] Tsungnan, L., Wang, C., Lin, P.-C. A neural-network-based context-aware handoff algorithm for multimedia computing. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP) archive*, vol 4, issue 3, 2008, pp 1-23.

[Va08] Vainio, A.-M. Proactive Fuzzy Control and Adaptation Methods for Smart Homes. *IEEE Intelligent Systems*, March 2008, pp. 42-49.

[VP06] Vasilakos, A. and Pedrycz, W. *Ambient Intelligence, Wireless Networking, and Ubiquitous Computing*. Artech House Publishers, 2006.

[VS12] Venables, W.N., Smith, D.M. and the R Development Core Team. *An Introduction to R*, URL=http://cran.r-project.org/doc/manuals/R-intro.pdf, last accessed October, 30, 2012.

[VG04] Vintan, L., Gellert, A., Petzold, J. and Ungerer, T. Person Movement Prediction Using Neural Networks. in *First Workshop on Modeling and Retrieval of Context*, Ulm, Germany, September 2004.

[WZ04] Wang, X.H., Zhanupon, D.Q., Gu, T. and Pung, H.K. Ontology Based Context Modeling and Reasoning using OWL. in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, IEEE Computer Society, 2004, p. 18.

[We91] Weiser, M. The Computer for the 21stCentury. *Scientific American*, vol 265, no. 3, September 1991, pp 94–104.

[WG99] Weiser, M., Gold, R. and Brown, J.S. The origins of ubiquitous computing research at PARC in the late 1980s. *IBM Systems Journal*, vol. 38, Dec. 1999, pp. 693–696.

[WH07] Wilkin, G. A., Huang, X. K-Means Clustering Algorithms: Implementation and Comparison. in *Second International Multi-Symposiums on Computer and Computational Sciences (imsccs)*, pp.133-136, 2007.

[WM08] Williams, C.A., Mohammadian, A., Nelson, P.C. and Doherty, S.T. Mining sequential association rules for traveler context prediction. in *Proceedings of the 5th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, Dublin, Ireland: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 1-6.

[WG01] Wolf, J., Guensler, R. and Bachman, W. Elimination of the travel diary: An experiment to derive trip purpose from GPS travel data. in: *Notes from Transportation Research Board, 80th annual meeting*, Washington, D.C., 2001.

[WJ08] Wolfe, B., James, M.R. and Singh, S. Approximate predictive state representations. in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, Estoril, Portugal: International Foundation for Autonomous Agents and Multiagent Systems , vol. 1, 2008, pp. 363-370.

[Wr97] Wright, S. J. *Primal-Dual Interior-Point Methods.* SIAM, 1997.

[XP08] Xue, W., Pung, H., Ng, W. and Gu, T. Data Management for Context-Aware Computing. in *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing 2008 EUC '08*., 2008, pp. 492-498.

[YL08] Yager, R. R. and Liu, L. *Classic Works of the Dempster-Shafer Theory of Belief Functions.* Springer, 2008.

[YC11] Yan, Z., Chakraborty, D., Parent, C., Spaccapietra, S., and Aberer, K. SeMiTri: A Framework for Semantic Annotation of Heterogeneous Trajectories. in *14th International Conference on Extending Database Technology (EDBT '11)*, 2011, pp 259-270

[YW08] Yang, J.-Y., Wang, J.-S. and Chen, Y.-P. Using acceleration measurements for activity recognition: an effective learning algorithm for constructing neural classifiers. *Pattern Recognition Letter*, vol 29, issue 16, 2008, pp 2213–2220.

[YC07a] Ye, J., Coyle, L., Dobson, S. and Nixon, P. Using Situation Lattices to Model and Reason about Context. in *Proceedings of the Fourth International Workshop Modeling and Reasoning in Context (MRC 2007)*. Roskilde, Denmark, 2007.

[YC07b] Ye, J., Coyle, L., Dobson, S. and Nixon, P. Ontology-based models in pervasive computing systems. *Knowl. Eng. Rev.*, vol. 22, no. 4, Dec. 2007, pp. 315–347.

[YD12] Ye, J., Dobson, S. and McKeever. S. Situation identification techniques in pervasive computing: A review. *Pervasive Mob. Comput.* vol 8, issue 1, February 2012, pp 36-66.
DOI=10.1016/j.pmcj.2011.01.004
URL=http://dx.doi.org/10.1016/j.pmcj.2011.01.004, last accessed October, 30, 2012.

[YP92] Yeh, T.-Y. and Patt, Y.N. Alternative Implementations of Two-Level Adaptive Branch Prediction. in *Proceedings of the 19th International Symposium on Computer Architecture*, 1992, pp. 124-134.

[Za65] Zadeh, L.A. Fuzzy sets. *Information and control*, vol. 8, 1965, pp. 338–353.

[ZG10] Zhang, D., Guo, M., Zhou, J., Kang, D. and Cao, J. Context reasoning using extended evidence theory in pervasive computing environments. *Future Gener.*

Comput. Syst., vol 26, issue 2, 2010, pp 207–216.

[ZS11] Zhang, M. and Sawchuk, A. A. Context-aware fall detection using a Bayesian network. in *Proceedings of the 5th ACM International Workshop on Context-Awareness for Self-Managing Systems*, New York, NY, USA, 2011, pp 10–16.

[ZF04] Zhou, C., Frankowski, D., Ludford, P., Shekhar, S. and Terveen, L. Discovering personal gazetteers: An interactive clustering approach. in *Proc. ACMGIS*, 2004, pp 266–273.

[ZM08] Ziebart, B.D., Maas, A.L., Dey, A.K. and Bagnell, J.A. Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior. in *Proceedings of the 10th international conference on Ubiquitous computing*, 2008, pp 322–331.

[ZL78] Ziv, J. and Lempel, A. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, vol 24, issue 5, 1978, pp. 530–536.

[ZA99] Zukerman, I, Albrecht, D.W. and Nicholson, A.E. Predicting users' requests on the WWW. *Courses and Lectures-International Centre for Mechanical Sciences*, 1999, pp. 275–284.

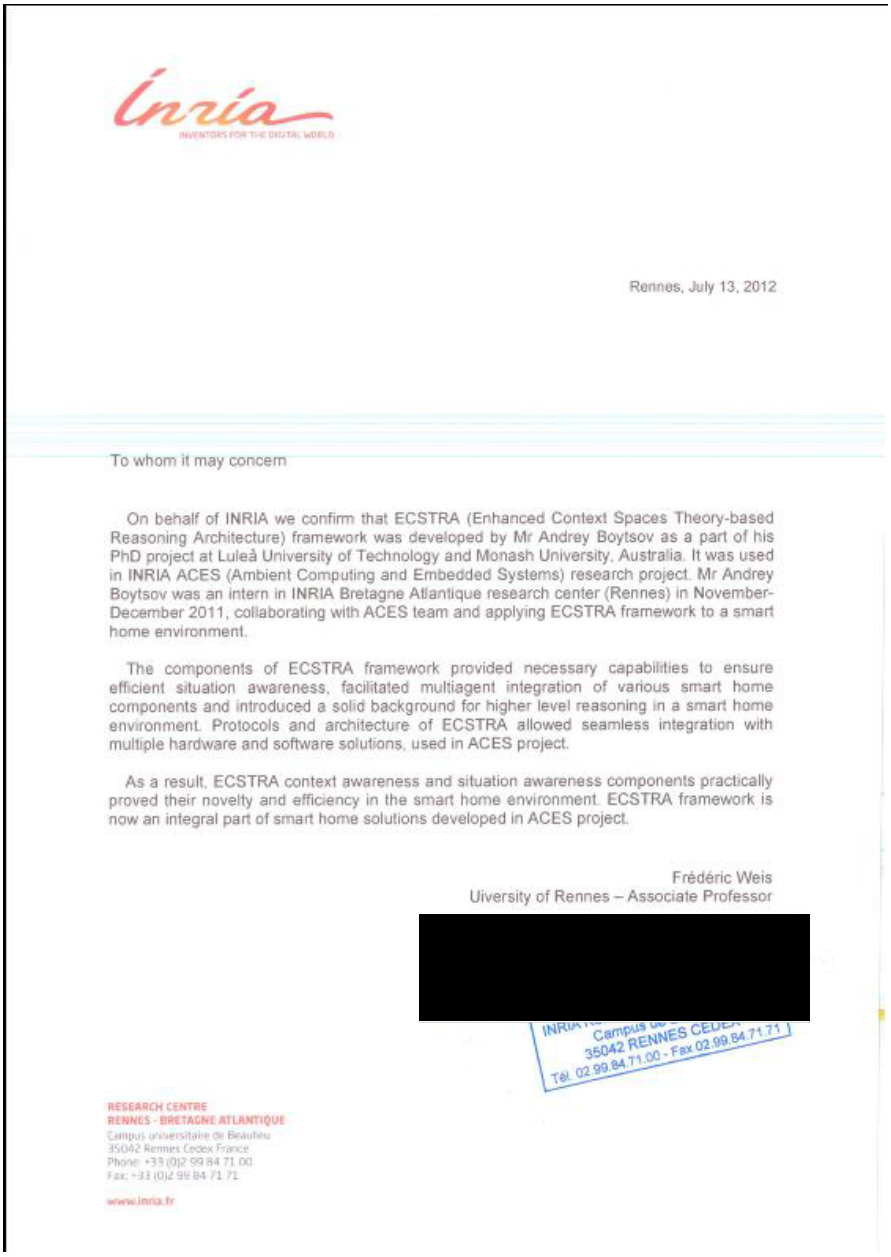
[Av11] *Avis Event Router Documentation*, URL=http://avis.sourceforge.net/documentation.html, last accessed October, 30, 2012.

[An12] *How to Build a Combined Agent Based / System Dynamics Model Logic in AnyLogic*, 2008. URL=http://www.xjtek.com/file/161, last accessed October, 30, 2012.

[El11] *Elvin Protocol Specifications*, URL=http://www.elvin.org/specs/index.html, last accessed October, 30, 2012.

[GP12] *Google Places API documentation*. URL=https://developers.google.com/maps/documentation/places/, accessed on October, 30, 2012.

[IS99] IST Advisory Group. *Information Society Technologies Advisory Group: Orientations for Workprogramme 2000 and Beyond*, pp. 3-4 (17. September 1999).



Rennes, July 13, 2012

To whom it may concern

On behalf of INRIA we confirm that ECSTRA (Enhanced Context Spaces Theory-based Reasoning Architecture) framework was developed by Mr Andrey Boytsov as a part of his PhD project at Luleå University of Technology and Monash University, Australia. It was used in INRIA ACES (Ambient Computing and Embedded Systems) research project. Mr Andrey Boytsov was an intern in INRIA Bretagne Atlantique research center (Rennes) in November-December 2011, collaborating with ACES team and applying ECSTRA framework to a smart home environment.

The components of ECSTRA framework provided necessary capabilities to ensure efficient situation awareness, facilitated multiagent integration of various smart home components and introduced a solid background for higher level reasoning in a smart home environment. Protocols and architecture of ECSTRA allowed seamless integration with multiple hardware and software solutions, used in ACES project.

As a result, ECSTRA context awareness and situation awareness components practically proved their novelty and efficiency in the smart home environment. ECSTRA framework is now an integral part of smart home solutions developed in ACES project.

Frédéric Weis
University of Rennes – Associate Professor

[redacted]

Campus
35042 RENNES CEDEX
Tél. 02.99.84.71.00 - Fax 02.99.84.71.71

RESEARCH CENTRE
RENNES - BRETAGNE ATLANTIQUE
Campus universitaire de Beaulieu
35042 Rennes Cedex France
Phone: +33 (0)2 99 84 71 00
Fax: +33 (0)2 99 84 71 71

www.inria.fr