Towards Engineering Models of Aspectual Pervasive Software Services

Dhaminda Abeywickrama
Clayton School of Information Technology
Monash University, VIC, Australia
dhaminda@infotech.monash.edu.au

Sita Ramakrishnan
Clayton School of Information Technology
Monash University, VIC, Australia
sitar@infotech.monash.edu.au

ABSTRACT
With the proliferation of ubiquitous computing devices and mobile internet it is envisaged that future pervasive services will be increasingly large-scale and operate at an inter-organizational level. Therefore, designing and implementing pervasive services will be a more complex and challenging task. Building software architectural models of concurrent and distributed pervasive services and their compositions provide engineers a better understanding of how these complex services inter-operate and help uncover any errors during the early stages of the software lifecycle. In this paper, we propose a novel approach based on behavioral modeling and analysis techniques for representing pervasive software services and their compositions and verifying process behavior of these models against specified system properties. As part of ongoing research, we model the crosscutting context-dependent behavior of our services as aspect-oriented models using a custom UML profile, and apply model transformation techniques to automatically translate the aspects into state-machine based behavioral representations to facilitate rigorous software process analysis. We explore our approach using an existing case study in transport and logistics.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *state diagrams.*

General Terms
Design, Verification.

Keywords
Context-aware service-oriented architectures, Aspect-oriented modeling, Model-driven architectures, Model-checking.

1. INTRODUCTION
Ubiquitous environments facilitate the collection of information from various data sources in order to aggregate the context of entities, such as users, places or objects. The context obtained from these sources can be used to automatically adapt a service's behavior or the content it processes to the context of one or several parameters of a target entity in a transparent way, resulting in context-aware services [10]. With the proliferation of ubiquitous computing devices and mobile internet, it is envisaged that future context-aware services will be large-scale and operate at an inter-organizational level, with an increasing number of actors and constraints involved [5]. Therefore, designing and implementing context-aware services will be more complex and challenging. There has been significant recent interest within the pervasive computing community for representing context-aware services at different stages of the software lifecycle. However, most of these efforts have concentrated on the design [2] or implementation stages [13] while little work has been done during the initial phase of design, such as architecture design.

The use of models has been a popular approach used by engineers when constructing complex systems. Behavior modeling and analysis have been successfully used by software engineers to uncover errors of concurrent and distributed systems at design-time. The concrete mathematical foundations exposed by behavioral models facilitate rigorous software process analysis and mechanical verification of properties using techniques, such as model-checking. In this paper, we propose a novel approach based on behavioral modeling and analysis techniques for modeling context-aware software services and their compositions and verifying the process behavior of these models against specified system properties. In particular, as part of our ongoing research, we model the crosscutting context-dependent functionality of the interacting pervasive services as aspect-oriented models using a custom UML profile. In order to facilitate rigorous software process analysis the context aspects are automatically translated to state machine based behavioral representations using transformation authoring of MDA [16]. The behavioral modeling approach presented in this paper for engineering aspectual pervasive services is particularly based on formal verification, validation and simulation techniques provided by the model-checker Labeled Transition System Analyzer (LTSA) [14] and its process calculus Finite State Processes (FSP) [14]. We explore our approach using a modified subset of a real-world case study in transport and logistics.

The rest of the paper is organized in the following sections. Section 2 provides an overview of the case study. In Section 3, we present our approach towards engineering aspectual pervasive services. A brief discussion on related work is provided in Section 4, and finally, Section 5 concludes the paper outlining future work.

2. TRANSPORT CASE STUDY
This research's approach is explored using a real-world case study in transport and logistics based on the ParcelCall project [7], a European Union project within the Information Society Technologies program. The case study describes a scalable, real-time, intelligent, end-to-end tracking and tracing system using Radio-Frequency Identification (RFID), sensor networks, and services for transport and logistics (Figure 1). This case study is particularly appealing to us as it provides several scenarios for

representing software services that inter-operate in a pervasive, mobile and distributed environment. A significant subset of the ParcelCall case study is exception handling that needs to be enforced when a transport item's context information exceeds or violates acceptable threshold values. This research is focused on this subset. The primary context parameters modeled in the study include item identity, location, temperature, pressure and acceleration.

3. SERVICE-ORIENTED DEVELOPMENT OF CONTEXT-AWARE SERVICES

3.1 Approach Overview

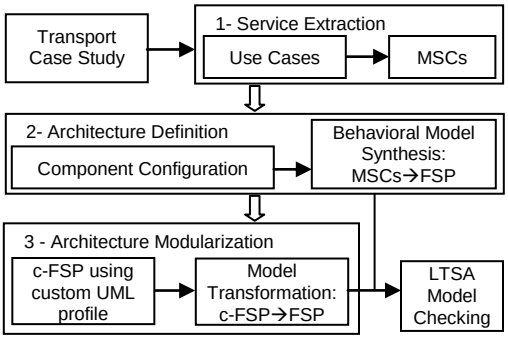

Figure 1. Pervasive service-oriented development process.

The overall pervasive service-oriented development process is divided into three main stages (Figure 1). Initially, using the case study subset we extract use cases and define a service specification for the system under consideration using message sequence charts. Second, the architecture for the system is defined using a component configuration and an architecture model in FSP. Third, the architecture model synthesized from previous step is modularized by proposing a new set of constructs called contextual-FSP (c-FSP) in a custom UML profile and translated into FSP before applying model-checking using the LTSA tool. We now elaborate the approach in detail.

3.2 Service Extraction

First, a service specification for the system under consideration is defined. The service extraction step is initiated by determining the relevant use cases of the case study subset. As stated in Section 2, this research is focused on the exception handling subset of the ParcelCall case study. Thus, the following use cases have been identified: monitor item location, generate alarm on item location, monitor item environment status, monitor temperature, monitor pressure, monitor acceleration, and generate alarm on item environment status. The identified use cases and their relationships are expressed using an UML use case diagram. After determining the use cases, we specify the services that realize the identified use cases. The term service is given different meanings by the software engineering community as it is used at various levels of abstraction. As defined by Kruger *et al.* [12], we identify a software service as the interaction pattern or the interplay of several components collaborating to complete a desired task. Furthermore, we also identify services as first-class modeling elements as opposed to first-class implementation elements, such as Web services. We use message sequence charts (MSCs) provided by the LTSA-MSC [19] tool to describe the interaction patterns defining the services. LTSA-MSC is an extension to the LTSA tool, which allows documenting scenarios (interaction patterns of services) in the form of MSCs and generating a behavioral model of the specification in FSP. MSCs have been a widely used notation to describe scenarios by software engineers. A MSC specification consists of several basic MSCs (bMSCs) and a high-level MSC (hMSC). A bMSC describes the interaction between a set of components while the hMSC provides hierarchy to the specification. In the study, the following software services (Figure 2) have been extracted and defined: interpret context, aggregate context, update database, observe events, broadcast, and parcelcall (composed service). Each service of the service repository is provided with a brief description, the components involved and a graphical representation of the interaction pattern that defines the service. However, for reasons of brevity in this paper we only provide the service definition for the broadcast service as provided next.

3.2.1 Broadcast Service

Description: The purpose of the Broadcast service is to alert the Mobile Device when an event is observed. The Controller commands the Notifier to broadcast a message to the Mobile Device. The Notifier relays the message to the Mobile Device. This service is triggered by the Observe Events service. *Components:* Controller, Notifier, Mobile Device. *Interaction:* Figure 2 (f).

3.3 Architecture Definition

Second, the architecture for the system under consideration is defined. To this end, first, we define a component configuration that implements the extracted services from Section 3.2 using an UML deployment diagram (Figure 3). The component configuration of the approach is based on a distributed version of the Observer pattern called the Event-Control-Action (E-C-A) architecture pattern [6]. Second, a behavioral representation of the service specification in the form of FSPs is generated automatically using the LTSA-MSC tool's FSP synthesis feature. This process essentially derives three behavioral models: an architecture model, a trace model and a constraint model. The architecture model provides the basis for modeling and reasoning about the system design where the components are modeled as LTSs. The overall system is effectively the parallel composition of all the components in the specification.

3.4 Architecture Modularization with c-FSP

Third, the architecture model synthesized in the previous step is modularized by applying separation of concerns. Separation of concerns is a design principle introduced by Dijkstra [8], which identifies the need to deal with issues one at a time. Context-handling information is considered to be tightly coupling (crosscutting) the core functionality of the service at service interface level. This results in a complex design, which is hard to implement and maintain. Therefore, in this paper we propose a custom UML profile called the c-FSP-UML profile to decouple the crosscutting context-dependent behavior of the services from the core service logic at service interface level. The context-handling constructs of the profile are collectively referred as the contextual-FSP (c-FSP) constructs. In the profile, we modularize context-dependent behavior of the services using aspect-oriented modeling (AOM). AOM is an aspect-oriented software

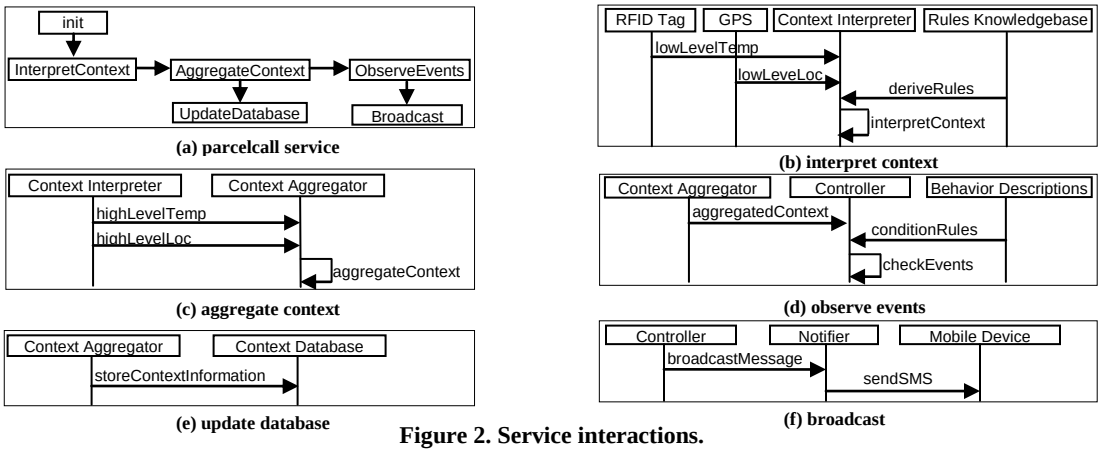

Figure 2. Service interactions.

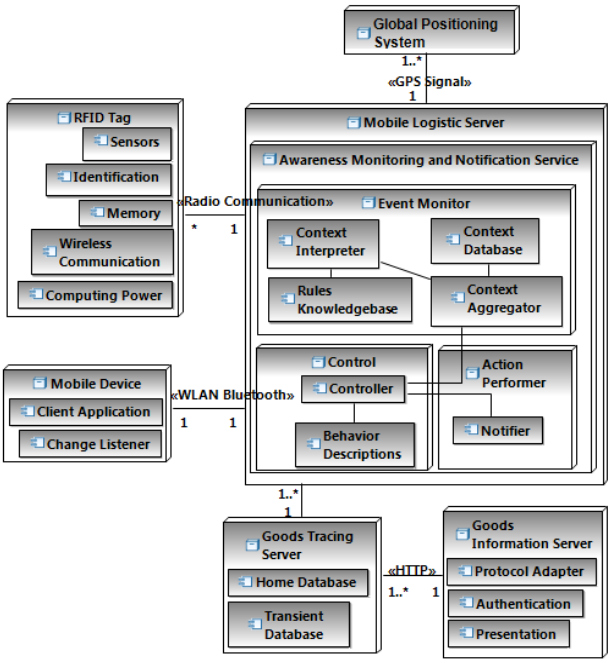

Figure 3. Architecture configuration.

development extension applied to the early stages of the software lifecycle to support separation of concerns at the modeling level. We use model transformation techniques to automatically translate the aspects of the profile (c-FSP aspects) to FSP semantics to facilitate rigorous process analysis by the LTSA tool.

3.4.1 The Notion of Context Applied

The notion of context used in this paper is based on a definition provided by Analyti *et al.* [3] for context in information modeling. The authors describe context as a set of objects, each of which is associated with a set of names and another context called its reference. Furthermore, the authors enhance the definition for context by stating that each object of a context is either a simple object or a link object (attribute, instance-of, ISA) and each object can be related to other objects through attribute, instance-of or ISA links. The authors use traditional object-oriented abstraction mechanisms of attribution, classification, generalization and encapsulation to structure the contents of a context.

3.4.2 The c-FSP-UML Profile

In this section, we present our custom UML profile (c-FSP-UML profile) proposed to decouple the crosscutting context-dependent functionality from the core service logic at service interface level (Figure 4). This profile provides a UML meta-level extension to the UML model [1] defined previously by the authors to separate the crosscutting context concerns. Using the profile we model core service logic and context-dependent behavior of a service as two separate concerns within the same model allowing the modification of context-dependent behavior of the service without affecting its main functionality. The c-FSP-UML profile encompasses constructs of both aspect-orientation and object-orientation (for example, attribution, classification, generalization and encapsulation) aimed at modularizing and reducing the complexity of context-dependent behavior at service interface level. As stated in Section 3.4, we modularize context-dependent behavior of services using aspect-oriented modeling. The service elements of the profile (service model) are represented by the state, transition, process, finite state machine, service and service specification classes while the aspect classes and the context source class represent the context elements (context model). The dependency relationships essentially associate service elements with context elements and context elements with their respective context sources. The precedence relationships are used for explicitly specifying how aspect precedence can be enforced to reduce the aspect interference problem at modeling level. The c-FSP constructs proposed in this paper essentially constitute the context elements, dependency relationships and precedence relationships of the profile. Brief descriptions of the c-FSP constructs are provided next.

- *Aspect, Advice, Pointcut classes:* Aspect class encapsulates several advices and pointcuts. Advice specifies the crosscutting behavior of the aspect while the Pointcut encapsulates a set of joinpoints. A joinpoint is the location (transitions) where the crosscutting behavior emerges in the service model. Three types of aspects are identified: context aspect, trigger aspect and recovery aspect.

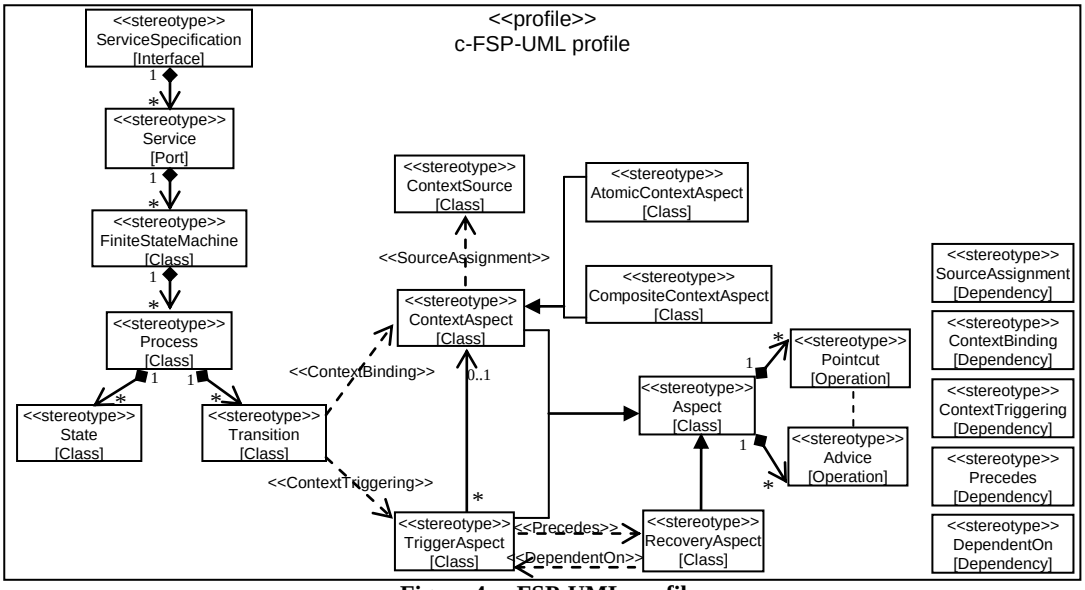

Figure 4. c-FSP-UML profile.

- *Context Aspect classes:* two types of context aspects are identified: atomic and composite. AtomicContextAspect class models low-level context readings from the context sources (for example, location or temperature) while CompositeContextAspect class encapsulates high-level derived context information (for example, isAdverseStatus). Also, we apply the notions of attribution, classification, generalization and encapsulation from the context definition (Section 3.4.1) to structure and link the objects defined in our aspects. Thus, we use both object-oriented and aspect-oriented notions to represent the context-dependent functionality of the services.

- *ContextSource class:* represents the resource from which context information is obtained, for example, RFID Tag and GPS.

- *TriggerAspect class:* models the contextual adaptation where the service is automatically executed or modified based on context information. For example, if isAdverseStatus is true then send a SMS to truck driver.

- *RecoveryAspect class:* models recovery actions that follow after an exception situation is raised by the trigger aspect. For example, control the refrigerator's temperature in the truck. This aspect is dependent on the Trigger aspect.

- *Dependency Relationships classes:* essentially associate service elements with context elements or context elements with their respective context sources. Three types of dependency relationships are identified: SourceAssignment, ContextBinding and ContextTriggering. SourceAssignment associates context attributes of a ContextAspect class with their respective context sources, which provide values for these attributes. ContextBinding models the automatic binding of service elements of the service model with context attributes of the ContextAspects class from the context model. ContextTriggering provides an association between service elements and triggering operations that may affect the service elements depending on context. Both ContextBinding and ContextTriggering dependency relationships essentially represent the binding of an aspect to its base class.

- *Precedence Relationships classes:* explicitly specify how aspect precedence can be enforced at the modeling level to reduce the aspect interference problem. Two types of precedence relationships are identified: Precedes and DependentOn. Precedes is used to indicate the precedence order for the aspects at a single joinpoint while DependentOn is used to specify that an aspect will only be matched on the existence of both aspects at the joinpoint.

3.4.3 Transformation of c-FSP Aspects to FSP

In this section, we briefly discuss our prototype tool - Aspectual FSP Generation Tool – proposed to automate the translation of c-FSP aspects to FSP in order to facilitate software process analysis by the LTSA tool. However, this step is work in progress. The tool is built using the IBM Rational Software Architect's (7.0) [11] transformation authoring feature, which is based on an Eclipse open-source project called Java Emitter Templates (JET). The prototype tool will effectively automate the transformation of c-FSP aspects (context, trigger, recovery aspects) of the UML models to textual FSP using the model-to-text transformation technique. The transformation is applied to UML models (class models and state machines) derived using the c-FSP-UML profile. Also, the stereotypes, properties and constraints specified in the profile can effectively be used in the transformation process. The transformation will be used to create infrastructure code or design pattern abstractions for the c-FSP aspects using FSP semantics. In general, an aspect in FSP needs to contain synchronization events (transitions) and waiting loops to coordinate with the base state machine and with other aspects. Also, each aspect type (context, trigger, recovery) contains its unique constructs which can be generated using model transformation. For examples, context aspect needs to contain constructs to read and write from the attribution, instance-of and ISA objects defined in the aspect. Trigger aspect requires constructs to alert and send notifications while recovery aspect needs constructs to recover from exception-handling situations. The creation of the prototype tool constitutes

several steps: create FSP exemplar files, perform exemplar analysis, define input schema and JET actions, edit JET templates, and finally execute the transformation with the user's input (Figure 5). These steps are briefly described next.

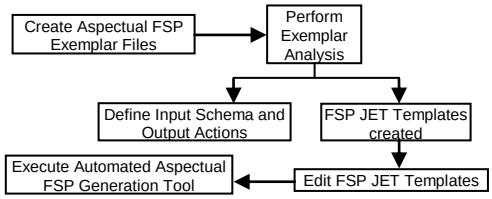

Figure 5. Tool for aspectual FSP generation.

- *Create FSP Exemplars for c-FSP Aspects:* Exemplars are representative examples of what the transformation needs to generate. An exemplar may contain one or more projects, files or folders, collectively referred to as exemplar artifacts. The creation of exemplar artifacts needs to be done following the best practices, guidelines and conventions, in order to get the best results when the transformation is eventually applied. In this research, we create FSP exemplars for the c-FSP aspects (context, trigger and recovery) of the UML model. After defining exemplars for the transformation an exemplar analysis project is created. This essentially creates an input schema and several JET templates, which initially are exact replicates of the FSP exemplars.

- *Define Input Schema and Output Actions:* In this step, the transformation input schema is defined which describes the structure of the transformation input. The input schema essentially defines the input the transformation expects (abstractions of the design pattern). Types and attributes can be added to the input schema to reflect the input that is required by the JET transformation. Also, output actions or JET actions need to be created and associated with the exemplar artifacts to demonstrate how the FSP artifacts are derived from the input schema (mapping of UML model elements with FSP aspects and semantics).

- *Edit JET Templates:* The JET templates for the FSP exemplars need to be edited so that they can adapt dynamically to accommodate the transformation input when the custom tool is executed. Editing JET templates may involve replacing text in the templates with references to the transformation input, or adding text to an iterating block or a conditional block.

- *Execute the Automated Aspectual-FSP Generation Tool:* After editing the JET templates the transformation can be executed by selecting the input to the transformation. The input to the transformation is our UML models (class models and state machines) created based on the c-FSP-UML profile. Using the tool the input model can be modified and FSP aspects can be generated dynamically in any Eclipse 3.2 environment.

For verification purposes, first, it is necessary to weave or compose the generated FSP for c-FSP aspects with their base state machines. This is discussed in the next section followed by the activities proposed for the actual verification using the LTSA tool.

3.4.4 Weaving Aspectual and Base State Machines

For verification purposes we first compose an aspectual state machine with its base state machine using an explicit weaving mechanism at the executable state machine level. The weaving process also helps the engineer to get a better understanding of the interacting services and their complex context-dependent behavior. The weaving is modeled as the parallel composition of the aspectual and the base state machines in FSP. The base state machine can contain multiple processes. An aspectual state machine can contain independent processes that run concurrently with the base state machine. Both the base and aspectual state machines contain synchronization events (transitions) to control their coordination and weaving order. The weaving essentially transforms the base and aspectual state machines into a new state machine which can be analyzed and simulated as a whole using the LTSA.

3.4.5 Concurrency and LTSA Model-Checking

We add additional concurrency and distributed notions to the interacting context-aware services and their compositions in the architecture model and apply formal model-checking techniques provided by the LTSA tool, such as safety, progress and absence of deadlocks, to verify and validate the process behavior of the services against specified system properties. A range of properties that extensively cover the system requirements are being formalized to be used in the verification process. The properties are expressed as safety and progress property processes. All behavioral (aspectual and base) and property processes are composed into a system-level process and fed into the LTSA for model-checking. The LTSA checks for property violations and if any violations are found it produces a counterexample which can be used to improve the state models or the system properties for the services.

4. RELATED WORK AND DISCUSSION

The work presented in this paper is related to several approaches as discussed in this section.

Model Driven Architecture (MDA) [16] is a framework defined by the OMG for software development which supports model-driven engineering of software systems. A key characteristic of MDA is automation which can be achieved using two main techniques: transformation and patterns. Model transformation automates the generation of artifacts from models. Transformation can be of three types: model-to-model, model-to-text (model-to-code), and refactoring. This research's approach applies model-to-text transformation to translate UML models into textual FSP.

ContextUML metamodel [18, 17] is an UML based modeling language proposed for model-driven development of context-aware Web services. The authors in [18, 17] demonstrate how UML can be used to specify information related to the design of context-aware services where context handling information is separated from the core service logic at service interface level. However, there are two main differences between their approach and this research's approach. First, the authors have taken no account of concurrency and distributed notions in their design or any formal verification aspects through techniques such as model-checking. Second, there is no application of aspect-oriented modeling in their profile as in this research's approach.

Douence *et al.* [9] propose a model for concurrent aspects which handles coordination issues between aspects and the base program and other aspects using the LTSA tool. The authors' approach is

similar to this research's approach as both approaches use models of concurrently executing aspects and base state machines using FSP semantics. However, our approach is largely different as it is based on context-aware services.

Previous approaches on the development of context-aware services have largely been at the design [2] or implementation stages [13, 15] of the software lifecycle. In [15], the authors propose an approach to include context in the composition of Web services. The authors present a generic approach where context is applied at different steps of service provisioning. Luo *et al.* [13] establish a framework that enables context-aware composition of Web services taking into account both the user's and the service's context when composing services. Almeida *et al.* [2] present a model-driven development approach to context-aware services consisting of three levels of models with different degrees of abstraction and platform independence. Thus, it is evident that most approaches are at the implementation or the design level. This research's approach is clearly distinctive in this respect as it is based at the software architectural level.

5. CONCLUSIONS

To summarize, in this paper we have proposed a novel approach for behavioral modeling and verification of software architectural models of context-aware services. Building architectural models of concurrent and distributed context-aware services and their compositions provide engineers a better understanding of how these complex services inter-operate and help uncover any errors during the early stages of the software lifecycle. The main contribution of this paper is a method of exploring the strengths of two levels of design abstractions - semi-informal UML meta-level extensions and formal finite state machines - for specifying context-dependent behavior of pervasive services. As for future work apart from the completion of the transformation and verification activities of the approach, we intend to investigate the possibility of generating FSP from a woven model (that is, perform model-level weaving) instead of the code-level weaving proposed at present in FSP.

6. REFERENCES

[1] Abeywickrama, D. and Ramakrishnan, S. 2008. A systematic approach to modeling and verifying context-aware services in SOAs, In *Proceedings of the 2008 International Workshop on Context-Aware Pervasive Communities: Infrastructures, Services and Applications* (Sydney, Australia, May 19, 2008). CAPC 2008. (To appear).

[2] Almeida, J. P. A., Iacob, M. E., Jonkers, H., and Quartel, D. 2006. Model-driven development of context-aware services, In *Proceedings of the 6th IFIP International Conference on Distributed Applications and Interoperable Systems* (Bologna, Italy, June 13-16, 2006). 213-227.

[3] Analyti, A., Theodorakis, M., Spyratos, N., and Constantopoulos, P. Contextualization as an independent abstraction mechanism for conceptual modeling, Information Systems Journal 32, 1 (March, 2007), 24-60.

[4] AOM Website. http://www.aspect-modeling.org/.

[5] Buchholz, T., Küpper, A., and Schiffers, M. 2003. Quality of context: what it is and why we need it, In *10th Workshop of the HP OpenView University Association* (Geneva, Switzerland, July, 2003). HPOVUA'03.

[6] Costa, P. D., Pires, L. F., and van Sinderen, M. 2005. Architectural patterns for context-aware services platforms, In *Proceedings of the Second International Workshop on Ubiquitous Computing* (Miami, USA, 2005). 3-19.

[7] Davie, A. Intelligent tagging for transport and logistics: the ParcelCall approach, Electronics & Communication Engineering Journal 14, 3 (June 2002), 122-128.

[8] Dijkstra, E.W. 1976. A Discipline of Programming. Prentice-Hall, Englewood Cliffs, New Jersey, USA.

[9] Douence, R., Le Botlan, D., Noyé, J., and Südholt, M. 2006. Concurrent aspects. In *Proceedings of the 5th international Conference on Generative Programming and Component Engineering* (Portland, Oregon, USA, October 22 - 26, 2006). GPCE '06. ACM, New York, NY, 79-88.

[10] Hegering, H. G., Küpper, A., Linnhoff-Popien, C., and Reiser, H. 2003. Management challenges of context-aware services in ubiquitous environments, In *Proceedings of the 14th IFIP/IEEE Workshop on Distributed Systems: Operations and Management* (Heidelberg, Germany, October 20-22, 2003). Springer-Verlag, 246-259.

[11] IBM Rational Software Architect Website. http://www.ibm.com/software/awdtools/architect/swarchitect.

[12] Krüger, I. H., Mathew, R., and Meisinger, M. 2006. Efficient exploration of service-oriented architectures using aspects, In *Proceeding of the 28th international Conference on Software Engineering* (Shanghai, China, May 20-28, 2006). 62-71.

[13] Luo, N., Yan, J., Liu, M., and Yang, S. 2006. Towards context-aware composition of web services, In *Proceedings of the Fifth International Conference on Grid and Cooperative Computing* (October 21-23, 2006). 494-499.

[14] Magee, J. and Kramer, J. 2006. Concurrency: state models & java programs (2nd Edition). John Wiley & Sons, Worldwide Series in Computer Science, April, 2006, 1-413.

[15] Mostefaoui, S. K. and Hirsbrunner, B. 2004. Context-aware service provisioning, In *Proceedings of the IEEE/ACS International Conference on Pervasive Services* (Beirut, Lebanon, July, 2004). 71-80.

[16] OMG Model Driven Architecture Website. http://www.omg.org/mda.

[17] Prezerakos, G. N., Tselikas, N. D., Cortese, G. 2007. Model-driven composition of context-aware web services using ContextUML and aspects, In *Proceedings of the IEEE International Conference on Web Services* (Salt Lake City, Utah, USA, July 9-13, 2007). ICWS 2007. 9-13.

[18] Sheng, Q. Z. and Benatallah, B. 2005. ContextUML: a UML-based modeling language for model-driven development of context-aware web services, In *Proceedings of the International Conference on Mobile Business* (Sydney, Australia, July 11-13, 2005). ICMB'05. 206-212.

[19] Uchitel, S., Kramer, J., and Magee, J. Synthesis of behavioral models from scenarios, IEEE Transactions on Software Engineering 29, 2 (2003), 99-115.