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 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 Unified Modeling Language (UML) profile, and apply model transformation techniques to automatically translate the aspects into state machine based behavioral representations to facilitate rigorous software process analysis. The approach is explored using an existing case study in transport and logistics.

Categories and Subject Descriptors

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 [9]. 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 [4]. 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 [12] 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 software process analysis the aspects are automatically translated to state machine based behavioral representations using transformation authoring of Model Driven Architecture (MDA) [15]. 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) [13] and its process calculus Finite State Processes (FSP) [13]. We explore our approach using a real-world case study in transport and logistics.

The rest of the paper is organized as follows. 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 [6], 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. 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 violates acceptable threshold values. This research is focused on this subset.

The reference scenario used in this research describes an awareness monitoring and notification service, which alerts exceptional situations that may arise on transport items primarily to the vehicle driver of the transport unit. The threshold values for environment status (e.g., temperature, pressure, acceleration) of transport items and route (location) for the vehicle are set by the carrier organization in advance. The awareness monitoring and notification service alerts the vehicle driver if items’ environment status exceeds acceptable levels or if an item is lost or stolen. 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

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. 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. [11], 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 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. In this study, the services extracted conform to the reference scenario described in Section 2. 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 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. [11], 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 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. In this study, the services extracted conform to the reference scenario described in Section 2. 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. 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 architecture pattern [5]. 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 labeled transition systems. The overall system is effectively the parallel composition of all the components in the specification.
Aspects

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 [7], 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 a 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 referred to as the c-FSP-UML profile to decouple the crosscutting context-dependent behavior of a service from the core service logic at service interface level. In the profile, we modularize the context-dependent behavior of a service using aspect-oriented models referred here as contextual-FSP aspects or c-FSP aspects. Aspect-oriented modeling, is an aspect-oriented software 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 c-FSP aspects of the profile to FSP semantics to facilitate process analysis by the LTSA tool.

3.4.1 The Notion of Context used

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 in [3] 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 discuss the 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 defined previously in our research work [1] to separate the crosscutting context concerns. The use of aspect-oriented models here further extends this work from the previous one, which was originally motivated by the ContextUML metamodel [17]. Using our profile we model the core service logic and the context-dependent behavior of a service as two separate concerns within the same model allowing the modification of the context-dependent behavior without affecting the main functionality. The core service logic of a service is represented by the State, Transition, Process, Service and ServiceSpecification classes while the rest of the classes represent the context-dependent functionality of a service. The c-FSP-UML profile encompasses constructs of both aspect-orientation and object-orientation aimed at modularizing and reducing the complexity of context-dependent behavior at service interface level. Brief descriptions of the different constructs of the profile are provided next with examples from the reference scenario.
The execution of this aspect is dependent on the existence of the example, control the refrigerator’s temperature in the vehicle unit. If an exception situation is raised by the trigger aspect, RecoveryAspect class: SMS to vehicle driver.

Classification, generalization and encapsulation from the context isAdverseStatus). Also, we apply the notions of attribution, encapsulates high-level derived context information (e.g., location or temperature) while CompositeContextAspect class models low-level context readings from the context sources (e.g., System (GPS)).

ContextAspect classes: two types of context aspects are identified: context aspect, trigger aspect and recovery aspect. These aspects are collectively referred to as the c-FSP aspects.

ContextSource class: represents the resource from which context information is obtained, e.g., RFID Tag or Global Positioning System (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 vehicle 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 vehicle unit. The execution of this aspect is dependent on the existence of the Trigger aspect.

Dependency Relationships classes: essentially associate the core service classes (service elements of the profile) with the context elements of the profile, or the context elements with their respective context sources. Dependency relationships are of three types. 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 with context attributes of the ContextAspects class. 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. Precedence relationships are of two types. 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) [10] 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 of the UML models to textual FSP using the model-to-text transformation technique. The UML models constitute the class models and state machines derived using the c-FSP-UML profile for the reference scenario. Also, the stereotypes, properties and constraints specified in the profile can be effectively 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, a context aspect needs to contain constructs to read and write from the attribution, instance-of and ISA objects defined in the
aspect. A trigger aspect requires constructs to alert and send notifications while a recovery aspect needs constructs to recover from exception-handling situations. The creation of the prototype tool contains 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.

- **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. In this research, we create FSP exemplars for the c-FSP aspects (context, trigger and recovery) of the UML models. 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 exemplars 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 aspects in FSP 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 as a whole using the LTSA.

### 3.4.5 Concurrency and 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 expressed as safety and progress property processes that extensively cover the system requirements are being formalized to be used in the verification process. 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) [15] is a framework defined by the Object Management Group 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 [17, 16] is an UML based modeling language proposed for model-driven development of context-aware Web services. The authors in [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. [8] 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 [12, 14] of the software lifecycle. In [14], 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. [12] 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 presented 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. In particular, in this paper, we have discussed our current efforts at exploring the strengths of both finite state machines and UML meta-level extensions for representing the context-dependent behavior of software services, and using model transformation techniques as a bridge between these two design abstractions. As for future work, we mainly intend to complete the transformation and verification activities of the approach as presented in this paper.

6. REFERENCES