Chapter 12
Context-Aware Services Engineering for Service-Oriented Architectures

Dhaminda B. Abeywickrama

Abstract With the proliferation of ubiquitous computing devices and the Internet, *context-aware Web services* continue to evolve from simple proof of concept implementations created in the laboratory to large and complex real-world services developed in industry. Context-awareness capabilities in service interfaces introduce additional challenges to the software engineer. In order to handle the additional complexities associated with these special services, *solid software engineering methodologies* are needed during their development and execution. This chapter proposes a novel software engineering-based approach, which leverages the benefits of model-driven architecture, aspect-oriented modeling, and formal model checking, for engineering context-aware services for service-oriented architectures. The approach has been validated using a real-world case study in intelligent transport. An evaluation framework has been established to validate the main methods and tools employed. We also present two key research directions, extending this work to further benefit the wider service engineering and pervasive computing communities.

12.1 Introduction

Web services are software components that can be distributed over standard internet technologies. They are designed to add interoperability between diverse, distributed and heterogeneous applications. A *context-aware Web service* is a special type of Web service that adapts its behavior or the content it processes to the context of one

The work reported here was performed during the author's Ph.D. studies. The author is currently affiliated with the University of Modena and Reggio Emilia, Italy.

D. B. Abeywickrama (✉)
Monash University, Clayton Campus, Wellington Road, Clayton, VIC 3800, Australia
e-mail: dhaminda.abeywickrama@gmail.com

or several parameters of a target entity in a transparent way (e.g., restaurant finder services) [20]. Context has been defined by Dey and Abowd ([16], p.3) as:

> any information that can be used to characterize the situation of an entity: an entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.

Context information is characterized by several qualities that make the development of context-aware services challenging compared to conventional Web services, such as a highly dynamic nature, real-time requirements, quality of context information and automation. The additional complexities associated with these special services necessitate the use of solid software engineering methodologies during their development and execution. Most state-of-the-art approaches to context-aware services relate to the detailed design or implementation stages [21, 27, 29] of the software life-cycle, such as context-aware Web services. Little work focuses on the early phase of design such as the software architectural level, thus providing motivation for our work.

This novel approach effectively leverages benefits of several software engineering principles such as *model-driven architecture*, separation of concerns through *aspect-oriented modeling*, and formal verification using *model checking*, for engineering context-aware software services. This research adopts model-driven development to represent complex crosscutting context-dependent functionality in service interfaces in a modular manner, and to automate the generation of state machine-based adaptive behavior. The crosscutting context-dependent information of the interacting pervasive services is modeled as aspect-oriented models in UML. Using automated model transformations, we ensure the correct separation of concerns of the crosscutting context-dependent functionality at both semi-formal UML modeling and formal behavioral specification levels. A prototype tool—`Aspectual FSP Generation`—applying an effective pipeline of model-to-model and model-to-text transformations has been built.

The generated context-dependent adaptive behavior and the core service behavior for the pervasive services are rigorously verified using formal model checking against specified system properties. Model checking is applied, first to check the behavior of the individual pervasive aspects and components, and second to verify the overall behavior of the woven model even if no errors are found in the individual aspects and components. These verification stages can be used to gain confidence before the complex pervasive services are actually implemented. The approach is explored using a real-world case study in intelligent transport. An evaluation framework is established to validate the main methods and tools developed.

This chapter discusses the key features of our overall research work [1, 6] for engineering context-aware services. Also, it presents two key research extensions extending this work to further benefit the wider service engineering and pervasive computing communities. Section 12.2 establishes the motivation for the current study. The case study used to validate our approach is presented in Sect. 12.3. In Sect. 12.4, a high-level description of the methodology proposed for engineering context-aware services is provided. Section 12.5 (models and transformations) and Sect. 12.6

(verification using model checking) address this process in detail. An evaluation framework to validate our work is provided in Sect. 12.7, and Sect. 12.8 discusses two key research extensions to this work. Section 12.9 concludes this chapter.

12.2 Related Work

Previous approaches to the development of context-aware services have largely been at the detailed design or implementation stages [21, 27, 29] of the software life-cycle. In [21], the authors have discussed an approach for the context-aware development of Web applications consisting of Web services. Application modeling has been performed in UML, and a composite Web application targeting different implementation platforms has been generated. Context adaptation takes place on top of the Web application business functionality. Serral et al. [27] introduced a model-driven development method for developing context-aware pervasive systems. A context-aware pervasive system has been specified using a set of models, and automated code generation has been used to generate system Java code. Service adaptation has been performed using an Web Ontology Language specification.

Two main limitations can be identified on state-of-the-art approaches on context-aware services development. First, most existing approaches to representing context-aware services focus on the detailed design or implementation phases of the software life-cycle such as context-aware Web services. Little attention has been given to the early phase of design such as the software architectural level. Building software architectural models of pervasive services provides engineers with a better understanding of how these complex services interoperate and helps uncover any errors during the early stages of the software life-cycle. Second, most of the existing work applies the software engineering techniques of model-driven architecture [7, 9, 28], aspect-oriented modeling [17, 32], and formal model checking [12] in isolation and does not explore the combination of these technologies in the same approach. For example, Sheng and Benatallah [28] have taken no account of any formal verification aspects through techniques such as model checking nor have they applied aspect-oriented modeling in their UML profile. The integration or the synergy of these sound software engineering techniques would mutually complement and augment each other if used in a single approach. While the application of these techniques in isolation can be found in existing work in service engineering, however, an integrated architecture-centric solution aimed at managing the complexities associated with context-aware services is novel, as proposed in this chapter.

12.3 Case Study: Intelligent Transport

This section describes the case study that is used to validate our approach, and the notion of context applied in our work.

The research approach is explored using a real-world case study in intelligent tagging for transport known as the ParcelCall project [14]. 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 the current research as it provides several scenarios for representing software services that interoperate 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. The reference scenario used in this research describes an `awareness monitoring and notification pervasive service`, which alerts with regards to any 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 service alerts if items' environment status exceeds acceptable levels or if an item is lost or stolen during transport. The primary context parameters modeled in the study include item identity, location, temperature, pressure and acceleration.

The *notion of context* used in our work is based on a definition provided in [8] for context in information modeling. The authors in [8] describe context as a set of objects, each of which is associated with a set of names and another context called its reference. Furthermore, they 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. They use traditional object-oriented abstraction mechanisms of attribution, classification, generalization and encapsulation to structure the contents of a context.

12.4 Context-Aware Services Engineering Process

In this section, we introduce the software engineering process followed for generating context-dependent adaptive behavior for pervasive services (see Fig. 12.1) [3]. Sections 12.5 (models and transformations) and 12.6 (verification) address this further.

The current approach is particularly based on (i) the model-driven development techniques provided by the IBM Rational Software Architect [15], (ii) the formal verification techniques provided by the model checker Labeled Transition System Analyzer (LTSA) [23] and its process calculus Finite State Processes (FSP), and (iii) the LTSA tool's message sequence charts extension (LTSA-MSC). Java Emitter Templates (JET) is an open-source technology developed by IBM. JET is included in IBM Rational Software Architect and it is typically used in the implementation of a code generator [15]. One of the main objectives of the current research is to perform rigorous verification of the pervasive specification using formal model checking. Therefore, we use finite state machines as opposed to other formalisms such as petri-net based models.

The model transformation tool created in this study for adaptive behavior generation is called the `Aspectual FSP Generation tool`. The crosscutting

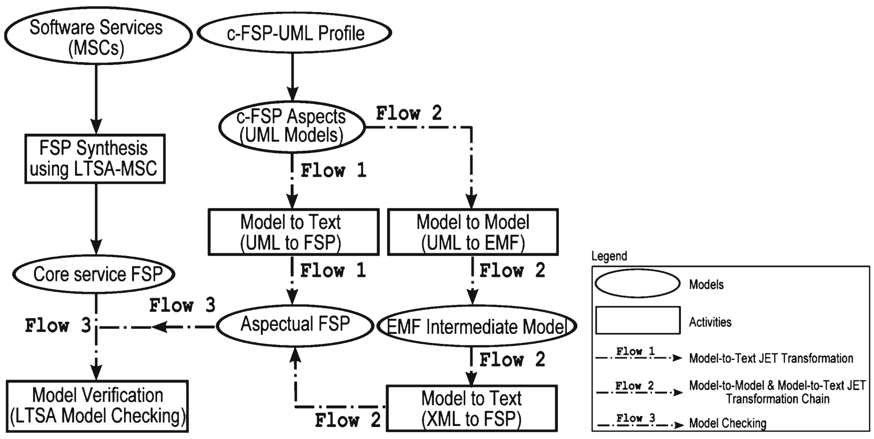

Software Services (MSCs)
c-FSP-UML Profile
c-FSP Aspects (UML Models)
Flow 2
Flow 1
FSP Synthesis using LTSA-MSC
Model to Text (UML to FSP)
Model to Model (UML to EMF)
Core service FSP
Flow 1
Flow 2
Flow 3
Flow 3
Aspectual FSP
EMF Intermediate Model
Flow 2
Model Verification (LTSA Model Checking)
Flow 2
Model to Text (XML to FSP)
Legend
Models
Activities
Flow 1 Model-to-Text JET Transformation
Flow 2 Model-to-Model & Model-to-Text JET Transformation Chain
Flow 3 Model Checking

Fig. 12.1 Context-aware services engineering process [3]

context-dependent information of the interacting pervasive services is modeled as aspect-oriented models in UML (`contextual-FSP aspects` or `c-FSP aspects`). Research works related to the aspect-oriented paradigm include composition filters, subject-oriented programming, adaptive programming, multi-dimensional separation of concerns and generative programming. However, compared to these approaches the aspect-oriented paradigm provides better language and tool support, and thus it is widely used in the software engineering community.

We use model transformations to automate the application of design patterns and generate infrastructure code for the `c-FSP aspects` using FSP semantics. The current study explores the strengths of both semi-formal UML meta-level extensions and formal finite state machines for representing the context-dependent behavior of software services, and model transformation techniques are applied as a bridge to enforce correct separation of concerns between these two design abstractions. The main benefits of this approach are: improving the quality and productivity of service development; easing system maintenance and evolution; and increasing the portability of the service design for the pervasive services engineer.

This approach focuses on the application of model-driven development for engineering pervasive services at finite state machine level. An aspect in FSP can be identified as an independent finite state machine that executes concurrently and synchronizes with its base state machine. In general, an aspect in FSP needs to contain synchronization events (transitions) to coordinate with its base state machine and other aspects. Also, each aspect type (e.g., `context`, `trigger` and `recovery`) contains its unique constructs which can be generated automatically using model transformation techniques. For example, a `trigger aspect` requires constructs to alert and send notifications while a `recovery aspect` needs constructs to recover from exception-handling situations. On the other hand, a `context aspect` has attribution, instance-of, ISA and reference constructs from the notion of context applied in this research.

In Fig. 12.1, the models and activities of the engineering process are represented as ellipses and square boxes respectively. The overall engineering process is structured into three main flows of activities. Both `Flow 1` and `Flow 2` originate from the `c-FSP-UML profile`. This profile effectively describes our conceptual model for context-dependent adaptive behavior using the aspect-oriented modeling paradigm. Using the profile we derive a UML model template and a UML class model to be used in the transformations, which are elaborated in Sect. 12.5.1. Two variations of the `Aspectual FSP Generation tool` have been built, which are represented using `Flow 1` and `Flow 2` in Fig. 12.1. Initially, a model-to-text JET transformation (`Flow 1`) was implemented with XPath expressions to navigate the UML class model for the `c-FSP aspects` and extract model information dynamically to the transformation. However, JET's support for UML models has several limitations. Therefore, a more effective solution was implemented as shown by `Flow 2`, which contains an effective pipeline of model-to-model and model-to-text JET transformations. The details of this transformation and its benefits are discussed in Sect. 12.5.2. The LTSA-MSC tool has been used to generate the architecture model in FSP for the service specification from which the core service model was extracted. `Flow 3` contains activities for rigorously verifying the models generated for the core service behavior and the context-dependent adaptive behavior using formal model checking (see Sect. 12.6).

12.5 Models and Transformations

12.5.1 Models: c-FSP-UML Profile and c-FSP Aspects

This subsection elaborates on the `c-FSP-UML profile` and the UML class model created with `c-FSP aspects` to modularize the service architecture (see Figs. 12.2, 12.3) [3].

Using the `c-FSP-UML 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`, `FiniteStateProcess`, `Service` and `Service Specification` classes while the rest of the classes represent the context-dependent functionality. 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 the service interface level. The use of aspect-oriented modeling in the profile further extends the UML model [2] created previously, which was originally motivated by the ContextUML metamodel [28].

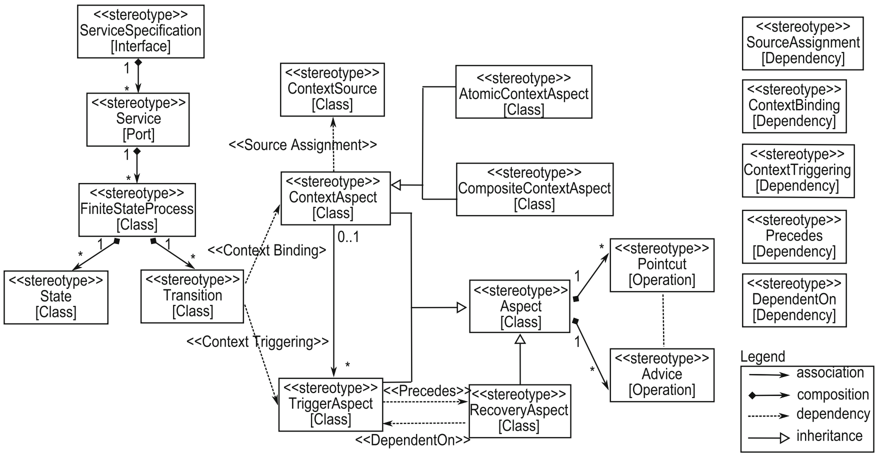

<<stereotype>> ServiceSpecification [Interface]
<<stereotype>> Service [Port]
<<stereotype>> FiniteStateProcess [Class]
<<stereotype>> State [Class]
<<stereotype>> Transition [Class]
<<stereotype>> ContextSource [Class]
<<Source Assignment>>
<<stereotype>> ContextAspect [Class]
<<Context Binding>>
<<Context Triggering>>
<<stereotype>> AtomicContextAspect [Class]
<<stereotype>> CompositeContextAspect [Class]
<<stereotype>> Aspect [Class]
<<stereotype>> Pointcut [Operation]
<<stereotype>> Advice [Operation]
<<stereotype>> TriggerAspect [Class]
<<Precedes>>
<<stereotype>> RecoveryAspect [Class]
<<DependentOn>>
<<stereotype>> SourceAssignment [Dependency]
<<stereotype>> ContextBinding [Dependency]
<<stereotype>> ContextTriggering [Dependency]
<<stereotype>> Precedes [Dependency]
<<stereotype>> DependentOn [Dependency]
Legend
association
composition
dependency
inheritance

Fig. 12.2 c-FSP-UML profile [3]

The aspect-oriented UML class model provided (see Fig. 12.3) contains several classes on the case study with stereotypes applied from the `c-FSP-UML profile`. As in the profile, this UML class model contains several constructs for representing the core service behavior, the context-dependent adaptive behavior, and the dependencies between the core service model and the context-dependent model. The core service behavior of the model is represented by classes, such as `ParcelCall`, `InterpretContext`, `Broadcast`, `Recovery`, `ObserveEvents`, the `FiniteStateProcess classes`, and the `Transition classes`. The `c-FSP aspects` of the UML class model are represented by several classes,

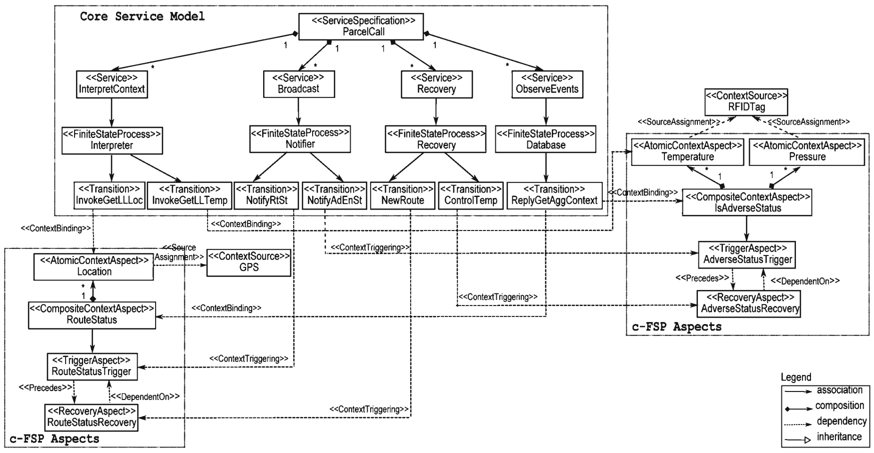

Core Service Model
<<ServiceSpecification>> ParcelCall
<<Service>> InterpretContext
<<Service>> Broadcast
<<Service>> Recovery
<<Service>> ObserveEvents
<<FiniteStateProcess>> Interpreter
<<FiniteStateProcess>> Notifier
<<FiniteStateProcess>> Recovery
<<FiniteStateProcess>> Database
<<Transition>> InvokeGetLLLoc
<<Transition>> InvokeGetLLTemp
<<Transition>> NotifyRtSt
<<Transition>> NotifyAdEnSt
<<Transition>> NewRoute
<<Transition>> ControlTemp
<<Transition>> ReplyGetAggContext
<<ContextSource>> RFIDTag
<<SourceAssignment>>
<<AtomicContextAspect>> Temperature
<<AtomicContextAspect>> Pressure
<<ContextBinding>>
<<CompositeContextAspect>> IsAdverseStatus
<<TriggerAspect>> AdverseStatusTrigger
<<Precedes>>
<<DependentOn>>
<<RecoveryAspect>> AdverseStatusRecovery
c-FSP Aspects
<<ContextTriggering>>
<<AtomicContextAspect>> Location
<<Source Assignment>>
<<ContextSource>> GPS
<<CompositeContextAspect>> RouteStatus
<<TriggerAspect>> RouteStatusTrigger
<<RecoveryAspect>> RouteStatusRecovery
Legend
association
composition
dependency
inheritance

Fig. 12.3 UML class model derived from the profile with c-FSP aspects

such as `Temperature`, `Pressure`, `IsAdverseStatus`, `AdverseStatusTrigger` and `AdverseStatusRecovery`. The different constructs of the profile and the UML class model are described next with examples from the case study.

- The `Aspect class` encapsulates several advices and pointcuts. The `Advice` specifies the crosscutting behavior of the aspect while the `Pointcut` encapsulates a set of joinpoints. A joinpoint is the location (transition) where the crosscutting behavior emerges in the service model. Three types of `c-FSP aspects` are identified: `context`, `trigger` and `recovery`.
- Two types of `context aspects` are identified: `atomic` and `composite`. `AtomicContextAspect class` models low-level context readings from the context sources (e.g., `Location` or `Temperature` in Fig. 12.3) while `CompositeContextAspect class` encapsulates high-level derived context information (e.g., `IsAdverseStatus` or `RouteStatus`). Also, the notions of `attribution`, `classification`, `generalization` and `encapsulation` from the context definition are applied to structure and link the objects defined in the aspects. Thus, both object-oriented and aspect-oriented notions are used to represent the complex context-dependent functionality of the services.
- The `ContextSource class` represents the resource from which context information is obtained, for example, `RFID Tag` or `GPS`.
- The `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 an SMS to vehicle driver (`AdverseStatusTrigger`).
- The `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 (`AdverseStatusRecovery` in Fig. 12.3).
- `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. There are three types of dependency relationships. `SourceAssignment` associates the context attributes of a `ContextAspect class` with their respective context sources, which provide values for these attributes. For example, the `SourceAssignment` relationship associates the `Location aspect` and the `GPS`, which provides GPS coordinates to the aspect (Fig. 12.3). `ContextBinding` models the automatic binding of service elements with context attributes of the `ContextAspect class`. `ContextTriggering` provides an association between service elements and triggering operations that may affect the service elements depending on context. In the case study this can be, for example, the association between the `NewRoute` transition of the core service model and the `RouteStatusRecovery aspect` of the context model (Fig. 12.3). 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. There are two types of precedence relationships. `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. For example, as shown in Fig. 12.3, the `AdverseStatusRecovery aspect` is executed following the `AdverseStatusTrigger aspect` (`Precedes`), and the existence of both these aspects are required at the joinpoint (`DependentOn`).

12.5.2 Model Transformations

The previous subsection discussed the `c-FSP-UML profile` and aspect-oriented models in UML (`c-FSP aspects`) derived to modularize the service architecture. The current subsection provides the model transformations created to automate the transformation of those UML models to formal behavioral specifications in FSP.

This multi-stage transformation chain describes an effective pipeline of model-to-model and model-to-text JET transformations (see Fig. 12.1, `Flow 2`) [3]. In this solution, first a model-to-model mapping transformation is created which extracts relevant information from the UML model elements and stereotypes, and then a code generator specific Eclipse Modeling Framework (EMF) intermediate model is built which contains only information required for the back-end model-to-text JET transformation. The front-end model-to-model transformation automatically invokes the back-end JET transformation. The benefits of this transformation are:

- As the JET transformation is independent of UML, UML expertise is no longer a requirement for the transformations [15]. This effective multi-stage transformations approach permits the development and validation of pattern implementations independent of any complexities associated with the UML metamodel.
- The JET transformation can be automatically invoked by the front-end model-to-model transformation. Therefore, the software engineer does not have to see or be aware of the back-end JET transformation.

The above factors can make the pattern implementation process a more accessible solution to the software engineer. As a result, this multi-stage transformations approach has been employed in the current research to transform the `c-FSP aspects` into formal FSP code, which has several steps.

- *Build an Intermediate EMF Model*. In this study, an EMF project is created as the intermediate model to be used in the transformations. The intermediate model can be based on EMF or XML. However, EMF has a rich metamodel and with EMF a Java API for the model can be generated [15].
- *Build a UML Profile*. The `c-FSP-UML profile` discussed earlier is used to augment the standard UML with information that is necessary for generation of

aspects in FSP. This profile defines stereotypes identifying core service elements, context-dependent information and dependencies for the pervasive software services. Stereotypes provide an efficient mechanism for extending the information that is stored on UML model elements. As any changes to the stereotypes in the profile affect the underlying UML model elements, it is important to track the profile version. To this effect, the `c-FSP-UML profile` is released. The profile can be distributed by creating an Eclipse plug-in that publishes the profile. To this end, the existing `c-FSP-UML profile` project is converted into a plug-in project. By publishing the profile as an Eclipse plug-in, any Eclipse-based product that installs this plug-in has access to the c-FSP-UML profile and its stereotypes.

- *Build a Model-to-Model Mapping Transformation.* After creating the `c-FSP-UML profile` and the intermediate EMF model, next we create a model-to-model mapping transformation (`aspectsFrontendMap`) that effectively maps the profile applied UML class model for the `c-FSP aspects` with the EMF intermediate model. This mapping transformation essentially associates elements of the input model (UML class model with the `c-FSP aspects`) with elements of the output model, which is the EMF intermediate model (`aspectsEMFModel`). To this end, several types of maps have been created. A map defines how data from an input type (e.g., a UML class) are copied to an output model type (e.g., an `aspect`). A map can move data from a source element to a target element using three methods: move transformations, custom transformations and submap transformations. This study uses a combination of *move* and *submap* transformations to associate elements of the UML class model created for `c-FSP aspects` with the elements of the EMF intermediate model (`aspectsEMFModel`). In this study, the following maps have been created: (i) UML Model elements to `Root` elements; (ii) UML Package elements to `AspectPackage` elements; (iii) UML Class elements to `Aspect` elements; (iv) UML Operation elements to aspect's `Operation` elements; and (v) UML Property elements to aspect's `Property` elements. After creating the mappings, the transformation source code is generated and customized to invoke the back-end model-to-text JET transformation automatically from the front-end model-to-model transformation. To link the model-to-model transformation's output to the back-end model-to-text JET transformation, the TransformationProvider Java file needs to be edited with the ID of the transformation to invoke.
- *Debug and Test.* Finally, we test the transformations created to verify that they are correct and function as required. This involves setting up a test environment called a *run-time workbench*, in which the plug-ins created are installed. Testing using a run-time workbench effectively launches a second copy of the Eclipse-based product. Testing the transformations involves: build the UML class model with `c-FSP aspects` and apply the `c-FSP-UML profile` to it; create a transformation configuration and execute the transformation configuration. In the transformations created in this study, the aspect name in UML becomes the process (state machine) name in FSP while operations and properties for the aspect in UML are used for generating states and transitions of the aspectual state machine in FSP. This represents the variable nature (point of variability) of the transformations.

Other than that, as stated previously, the transformations generate infrastructure (skeleton) code for the aspects.

Next the generated context-dependent adaptive behavior and the core service behavior for the pervasive services are rigorously verified using formal model checking against specified system properties.

12.6 Formal Verification

As discussed in Sect. 12.5.1, the crosscutting context-dependent behavior in service interfaces has been modeled using aspect-oriented UML models. To this effect, a custom UML profile (`c-FSP-UML profile`), a UML model template and UML class models (`c-FSP aspects`) have been created to modularize context information with several stereotypes. UML has been a widely applied technique for modeling object-oriented design or core design of a software specification. Also, exploring the meta-level notation of UML or extending the UML notation has been a popular approach used by many researchers for specifying crosscutting concerns. However, one of the main limitations of UML is its lack of support for rigorous verification due to its informal or semi-formal nature.

The expressive power of aspects in design specifications can be potentially harmful. The *crosscutting* nature and the *obliviousness* principle of aspects are two main issues that can introduce an additional correctness problem in an aspect-oriented design specification. These can create several problems or risks such as partial weaving, unknown aspect assumptions, unintended aspect effects, arbitrary aspect precedence, failure to preserve state invariants, and incorrect changes in control dependencies [24, 25]. Therefore, in order to address the main challenges associated with aspect-oriented modeling in software specifications (i.e., the semi-formal nature of UML notations and the expressive power of aspects), tool support such as automatic model checking is highly desirable to ensure the correctness of the specification.

12.6.1 Model Checking Aspectual Pervasive Software Services

In this subsection, we provide an overview of our approach for rigorously verifying the models generated for the context-dependent adaptive behavior and the core service behavior using formal model checking [5] (see Fig. 12.1 (Flow 3), and Fig. 12.4).

The model checking process can be divided into three main tasks: *modeling* (Sects. 12.6.2–12.6.3), *specification* and *verification* (Sect. 12.6.4). Modeling is the task of converting the design into a formalism accepted by a model checking tool [11]. Specification is the stating of the properties that the design needs to satisfy, and verification is the actual validation of the models.

- *Modeling*. The modeling step involves two main tasks that are performed to obtain the context-dependent adaptive behavior and the core service model of the software services. In this study, the `Aspectual FSP Generation tool` is used to generate the context-dependent behavioral code in formal FSP. The LTSA-MSC tool is used to generate the architecture model for the service specification in FSP, which is used to extract the core service model of the services (see Figs. 12.1, 12.4). All service components and aspects are modeled as processes represented as finite state machines in FSP. To verify the pervasive service specification, first the aspects are woven into their base state machines in FSP using an explicit weaving mechanism. Then concurrency and distributed notions (see Sect. 12.6.3) are added to the service specification to facilitate reasoning by the LTSA tool. Abstraction mechanisms are introduced to reduce the size of the woven model.
- *Specification*. Properties provide a way of formalizing and verifying system requirements. Here the properties focus on the required effects of the pervasive aspects, service components and the woven model. Rigorous modeling and specification of properties are very important to identify any defects in the pervasive services early in the software life-cycle before these complex services are actually implemented. According to the system requirements from the case study subset, more than 30 properties have been formalized focusing on the required behavior

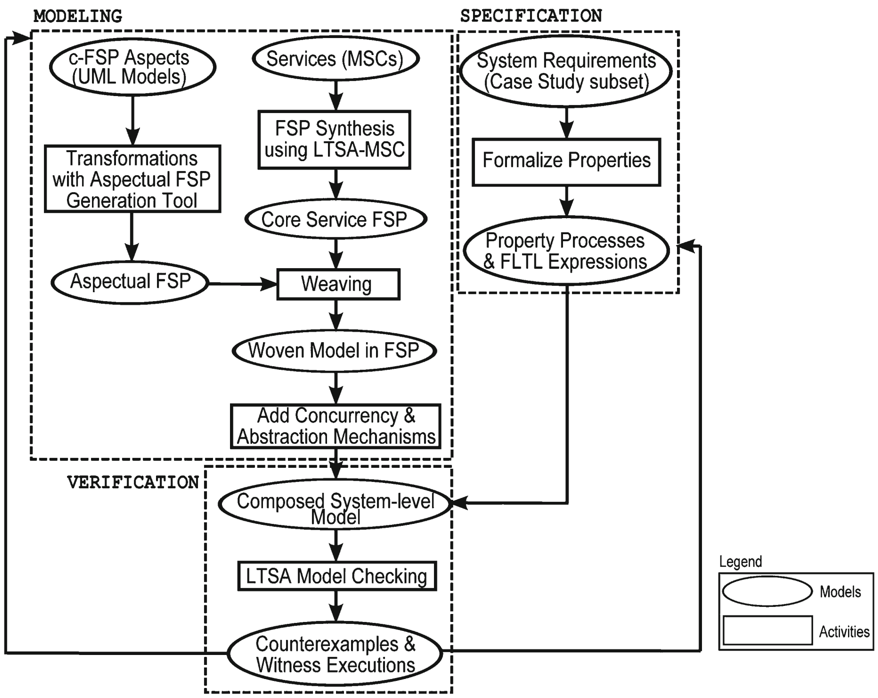

MODELING
SPECIFICATION
c-FSP Aspects (UML Models)
Services (MSCs)
System Requirements (Case Study subset)
FSP Synthesis using LTSA-MSC
Transformations with Aspectual FSP Generation Tool
Formalize Properties
Core Service FSP
Property Processes & FLTL Expressions
Aspectual FSP
Weaving
Woven Model in FSP
Add Concurrency & Abstraction Mechanisms
VERIFICATION
Composed System-level Model
LTSA Model Checking
Counterexamples & Witness Executions
Legend
Models
Activities

Fig. 12.4 Model checking aspectual pervasive software services [5]

from both service components and aspects. These properties have been expressed as property processes (safety and progress) and fluent linear temporal logic (FLTL) assertions.

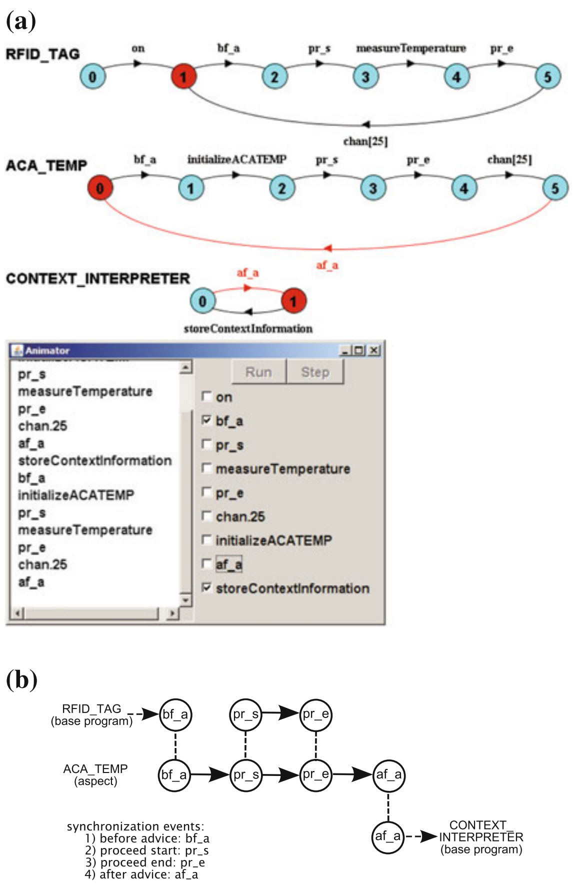
- *Verification*. Finally, all behavior and property processes are composed into a system-level process and this process is fed to the LTSA. The LTSA verifies whether any properties are violated and if so it reports a trace to the property violation known as a counterexample. Also, the use of FLTL assertions provides the opportunity to generate examples of traces (witness executions) which satisfy the property. The use of counterexamples and witness executions is exploited to identify and track any errors and their sources in the specification, which consists of several distributed service components and aspects collaborating with each other. Thus, this helps to iteratively improve the state models or the system properties for the aspectual pervasive software services.

12.6.2 Weaving of Pervasive Aspects and Components

Weaving of an aspect to its base state machine is important in order to analyze the overall system behavior. An explicit weaving mechanism is used here, where an aspect is woven into its base state machine using the *parallel composition operator* and *shared actions* in FSP. The main elements of the weaving process are the base program and an aspectual state machine (aspect). In general, the base program is not a single process but it is a combination of several processes. The base program (core service model) is specified as the parallel composition of the constituent base state machines. In order to support explicit parallel composition, the current study injects synchronization events in both the aspectual and base state machines. These events provide an effective mechanism to control the coordination between these state machines. The advice of an aspect contains three logical parts: `before advice` events, `proceed` events and `after advice` events. By using synchronization events the correct execution of these three sequences of actions with the base program can be ensured. Also, weaving of more than one aspect at the same joinpoint is possible using these explicit synchronization events.

The crosscutting elements of the joinpoint model and the weaving process are discussed next using a case study example. Figure 12.5a shows LTSs for three processes. The `RFID Tag` (`RFID_TAG`) and the `Context Interpreter` (`CONTEXT_INTERPRETER`) components are the base state machines while the `Atomic Context Aspect Temperature` (`ACA_TEMP`) is an aspectual state machine. The joinpoints of the base program are specified using the following synchronization events: `bf_a` (`before advice`), `pr_s` (`proceed start`), `pr_e` (`proceed end`), `af_a` (`after advice`). A pointcut is a sequence of joinpoints (i.e., the sequence of `bf_a`, `pr_s`, `pr_e` and `af_a`).

The execution and coordination of the base program and the aspect can be explained as follows (see Fig. 12.5b). The base program (`RFID_TAG`) emits the `bf_a` event to the aspect. The aspect performs an initialization operation



```
pr_s
measureTemperature
pr_e
chan.25
af_a
storeContextInformation
bf_a
initializeACATEMP
pr_s
measureTemperature
pr_e
chan.25
af_a
```

- [ ] on
- [x] bf_a
- [ ] pr_s
- [ ] measureTemperature
- [ ] pr_e
- [ ] chan.25
- [ ] initializeACATEMP
- [ ] af_a
- [x] storeContextInformation

Fig. 12.5 Weaving performed. **a** Weaving illustrated using LTSs. **b** Synchronization events

(`initializeACATEMP`), which is a before advice event. The base program waits for a control event from the aspect, which is a proceed event (`pr_s`) in this example. The base program performs the `measureTemperature` event and then emits `pr_e` to return the control back to the aspect. The aspect performs receiving of temperature readings using message passing, which is its after advice events. Finally, the base program (`Context_Interpreter`) waits for the end of advice event (`af_a`) from the aspect, and performs the `storeContextInformation` action. The woven program is modeled as the parallel composition of the base state machines and the aspect.

12.6.3 Concurrency Modeling

After weaving aspects into their base state machines the concurrency and distributed notions of the interacting pervasive software services are modeled to facilitate reasoning by the LTSA tool, such as message passing, shared objects and mutual exclusion (see Fig. 12.6).

The pervasive service specification includes several distributed service components and aspects collaborating with each other. These components and aspects encompass the active entities of the specification. It also includes shared objects and semaphores, which act as passive entities. All active and passive entities of the specification have been modeled as processes represented as finite state machines in FSP. In the specification, concurrency has been modeled using action interleaving.

This study models the `awareness monitoring and notification service` as a process-oriented context value chain (see Fig. 12.6). This value chain contains several stages: sensing, refinement, aggregation and contextualization. The context procurement and contextualization tasks of the pervasive service are driven by the `c-FSP aspects`. The communication between the distributed service components and aspects (e.g., between `RFID Tag` and `Atomic Context Aspect Temperature`) has been modeled using the synchronous message passing technique. The environmental readings (e.g., temperature, pressure) from the `RFID Tag` are sent using a single channel to the receiver (e.g., `Atomic Context Aspect Temperature`, `Atomic Context Aspect Pressure`) and the communication is one to one. In addition to using the message passing technique, shared objects have been used to model inter-process communication between the service components and the aspects. The problem of interference has been solved by enforcing mutually exclusive access to the shared objects. This has been modeled using binary semaphores, which are mechanisms for dealing with inter-process synchronization problems. For example, the `Atomic Context Aspect Temperature` aspect and the `Context Interpreter` component interact using a shared object for communicating temperature values used in the refinement stage of the context value chain of the pervasive service. The mutually exclusive access to this shared object has been enforced using a semaphore, thus only one process can access it at a given time. The `Context Database` process has been modeled as a shared resource where the `Context Interpreter` and the `Context Aggregator` service components write to it (writers) and the `Composite Context Aspect Route Status` and the `Composite Context Aspect Adverse Environment Status` aspects read from it (readers). This scenario has been modeled as a readers-writers problem with writers priority. The readers are denied access if there are writers waiting to acquire access and if a writer is not accessing the database any number of readers can access the database concurrently.

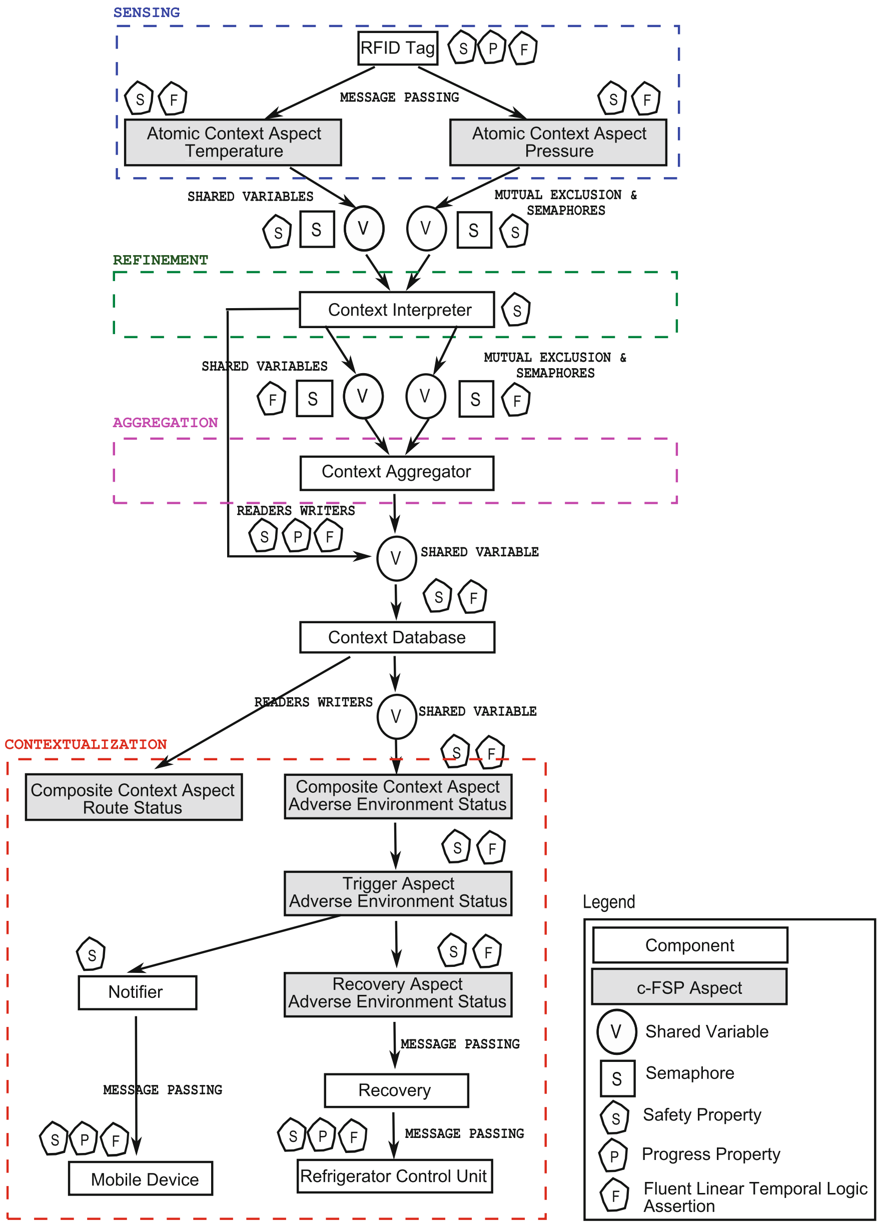


Fig. 12.6 Concurrency modeling between aspects and components [5]

12.6.3.1 Abstraction Mechanisms Applied

These are needed as a woven program may have too many states to be analyzed by the LTSA. One of the main challenges associated with model checking is the *state space explosion problem*. We use action hiding and minimization features available in FSP to reduce the size of the woven model before analyzing using the LTSA tool. For example, the actions modeled in the `Context Interpreter` and `Context Aggregator` components for enforcing mutually exclusive access to their shared variables are not required when modeling the readers-writers problem with writers priority, which involves the same components collaborating with aspects. Also, when executing the entire specification model, the partial order reduction feature has been used to reduce the size of the state space searched by the LTSA.

12.6.4 Properties Specification and Verification

Having discussed the modeling stage of the model checking process, the properties specification and verification stages are briefly addressed next (see [6, 5] for details). Properties have been expressed as property processes (safety and progress) and FLTL assertions.

Safety properties are used in a concurrent program to assert that nothing bad happens during the execution of the program [23]. In the case study subset, several safety properties have been specified for verifying (i) the behavior of the individual aspects and the components, and (ii) the overall behavior of the woven model even if no errors are found in the individual aspects and components. At the individual aspect or component level, a safety property has been defined for the `Trigger Aspect Adverse Environment Status` aspect to verify whether a notification is sent only when environment status is adverse. Another safety property has been defined for the `Context Interpreter` component to verify whether the refinement stage of the pervasive service is performed as expected. At the woven model level, safety properties have been defined to ensure the correct weaving of the base state machines and the aspectual state machines. These properties ensure that the ordering of the synchronization events is correct in the components and aspects of the woven models, thus ensuring the correct weaving of the components and the aspects at the joinpoints in the specification. For example, a safety property has been defined to ensure the correct weaving between the following components and aspects: `RFID Tag`, `Atomic Context Aspect Temperature`, `Atomic Context Aspect Pressure` and `Context Interpreter`. This property is composed with the woven process before performing analysis using the LTSA. LTSA analysis shows that there are no deadlocks or safety violations. Also at the woven model level, safety properties have been created to verify whether the mutually exclusive access to the shared variables is enforced properly.

Unlike safety properties, which are concerned with a program not reaching a bad state, liveness properties are concerned with a program eventually reaching a good state [23]. For example, in the case study subset, *progress properties* have been specified for the readers-writers problem. To this end, two progress properties have been defined to ensure that both readers (i.e., `Composite Context Aspect Adverse Environment Status`, `Composite Context Aspect Route Status` aspects) and writers (i.e., `Context Interpreter`, `Context Aggregator` service components) will eventually gain access to the lock to access the `Context Database` component. A progress analysis for this problem using the LTSA shows no errors.

In addition to safety and progress property processes, properties can be defined as state-based logical propositions in FSP. Fluents in FSP allow the expression of properties about the abstract state of a system at a particular point in time [23]. The current study employs *FLTL assertions* as a method for specifying system requirements of the case study subset. For example, two FLTL assertions have been defined to ensure mutually exclusive access to the shared variables by the `Context Interpreter` and the `Context Aggregator` service components. These properties ensure the required mutual exclusion safety property, and an additional liveness property, which asserts that if a process (i.e., `Context Interpreter` or `Context Aggregator`) enters the critical section that process should eventually exit before another process can enter. Verification performed for this logical property shows that there are no violations.

The use of FLTL assertions provides the opportunity to generate examples of traces (witness executions), which satisfy the property. This research applies *witness executions* as a means of identifying potential errors in the specification. For example, a FLTL property has been defined to verify the weaving of the base state machines and aspectual state machines in the specification. The negation of this assertion generates a counterexample. By using counterexamples and witness executions, the state models and system properties for the aspectual pervasive services are iteratively improved.

12.7 Evaluation Framework

This section provides the evaluation framework established to validate the research approach.

This evaluation framework [4] mainly validates the main contributions or deliverables of this study against several key evaluation criteria. The main tools used in this study include the `Aspectual FSP Generation tool` created in this research, the LTSA model checker and the LTSA-MSC tool. The method of evaluation is based on *key feature comparison*. Key feature comparison is used as a credible method for evaluating software engineering-based approaches [30]. The evaluation framework developed here does not produce additions to the research methodology but instead validates the methods and tools used in the research as a whole. The framework comprises a set of detailed criteria for two dimensions or views: *vertical*

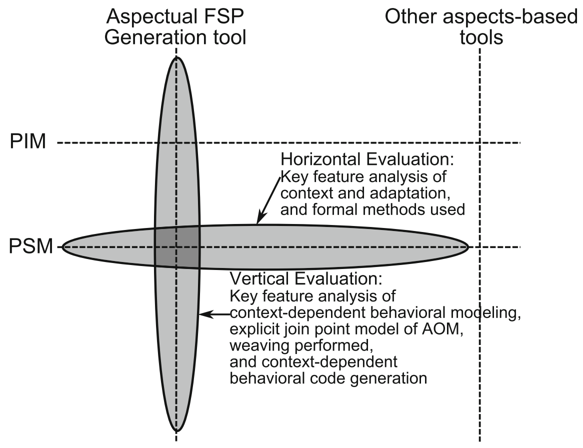

Aspectual FSP Generation tool
Other aspects-based tools
PIM
PSM
Horizontal Evaluation: Key feature analysis of context and adaptation, and formal methods used
Vertical Evaluation: Key feature analysis of context-dependent behavioral modeling, explicit join point model of AOM, weaving performed, and context-dependent behavioral code generation

Fig. 12.7 Evaluation framework: *vertical* and *horizontal* views

and *horizontal* (see Fig. 12.7). The notions of vertical and horizontal views were motivated by [30] (p. 14), which also uses a two-dimensional evaluation approach.

12.7.1 Vertical Evaluation of the Research

This evaluation focuses on comparing four tools across the modeling layers of platform-independent model (PIM) and platform-specific model (PSM) against the `Aspectual FSP Generation tool`. The compared tools are Groher and Schulze [19] approach, Whittle and Jayaraman [31] approach, Motorola WEAVR [13] and Fuentes et al. [18] approach (see [4] for details). Like the `Aspectual FSP Generation tool`, these tools have been developed using commercially available toolchains of similar area of application such as IBM Rational Software Modeler, Borland Together, Telelogic Modeller and Topcased [30]. This evaluation is based on the following criteria: context-dependent behavioral modeling at the PIM level, explicit joinpoint model of aspect-oriented modeling at the PIM level, weaving performed at the PIM or PSM level, and context-dependent behavioral code generation from the PIM to PSM level. A particular evaluation criterion can be fully satisfied (complete cover), partly satisfied (partial cover), or not supported at all.

The results of the vertical evaluation are assuring (see Table 12.1). Like the `Aspectual FSP Generation tool`, [13, 18, 19] support an explicit joinpoint model of aspect-oriented modeling at PIM level. Also, all the compared approaches support PIM or PSM level weaving of aspects. The vertical evaluation has demonstrated that the `Aspectual FSP Generation tool` has unique features on context-dependent behavioral modeling and context-dependent behavioral code generation. Table 12.1 shows that the `Aspectual FSP Generation tool` satisfies all the criteria as opposed to the other tools which satisfy only some criteria.

Table 12.1 Comparison matrix for vertical evaluation [4]

Evaluation criteria	Groher and Schulze	Whittle and Jayaraman	Cottenier et al.	Fuentes et al.	Aspectual FSP generation tool
PIM level support for context-dependent behavioral modeling	−	−	−	*	+
PIM level support for explicit joinpoint model of aspect-oriented modeling	+	−	+	+	+
PIM or PSM level support for weaving	+	+	+	+	+
PIM and PSM level support for context-dependent behavioral code generation	−	−	−	−	+

+ Complete cover of a criterion; * partial cover of a criterion; − no cover of a criterion

12.7.2 Horizontal Evaluation of the Research

In contrast to the vertical evaluation discussed above, the horizontal evaluation is aimed at investigating particular features of our approach at a single modeling level (i.e., the PSM level). These evaluation criteria cover two aspects of the study: the formal methods and tools employed in the study, and the context and adaptation dimensions of the customization approach used in the services.

12.7.2.1 Formal Methods and Tools Used in the Approach

Clarke et al. [10] provide several criteria that formal methods-based approaches and tools need to support. According to [10], although some of these criteria are ideals, it is still considered good to aim for them. The criteria are: early payback, incremental gain for incremental effort, multiple use, integrated use, ease of use, efficiency, ease of learning, orientation toward error detection, focused analysis and evolutionary development. The research methodology of the current study contains three stages: service specification, architecture definition and architecture modularization. In the present study, formal methods and tools (LTSA tool and LTSA-MSC tool) have been applied during the service specification and architecture definition stages of the research methodology, and finally for model checking the aspectual pervasive software services specification. This evaluates the application of the aforementioned formal methods and tools used in the current research against the criteria provided in [10]. Our approach has been evaluated using all the criteria provided by them (see [4]). However, due to space limitations this chapter discusses one key criterion. *Early* *payback*: this study is focused on the architectural level of the software life-cycle. This architecture-centric approach builds models of pervasive software services and their compositions and verifies their behavior against specified system properties. Building architectural models of pervasive software services allows the software engineers to validate the actual correctness of the services before the services are implemented later in the software life-cycle. Thus, it provides early payback or feedback to the service engineer on the validity of the services.

12.7.2.2 Context and Adaptation of the Customization Approach

Kappel et al. [22] and Schwinger et al. [26] present a comprehensive and uniform evaluation framework, which can be used to compare customization capabilities of approaches originating from the mobile computing and the personalization domains. The notion of customization refers to the adaptation of an applications services toward the current context. Their framework has two orthogonal dimensions, which are context and adaptation, and the mapping between context and adaptation has been represented by the notion of customization. They provide detailed criteria for both the context and adaptation dimensions of the framework. The *context* and *adaptation* dimensions of the customization approach used in the pervasive services of the current research are evaluated using those criteria. The results of this evaluation are summarized in two tables respectively: Tables 12.2 and 12.3. See [4] for a more detailed analysis of these results.

The horizontal evaluation of the approach has shown that the formal methods and tools employed in the research, and the customization approach used in the services, are effective toward the overall objectives of this research.

12.8 Research Extensions

In this section, we discuss two key research directions, extending this work to benefit the broader service engineering and pervasive computing communities.

12.8.1 Aspectual FSP Generation as an Integrated Eclipse Plug-in

The `Aspectual FSP Generation tool` developed in the current research can be extended as an integrated plug-in to the Eclipse development environment. This will be beneficial as it will allow our tool to be used in conjunction with other plug-ins for engineering context-aware services. Also, it can be leveraged by interested researchers in the wider service engineering community. To the best of our knowledge, such an integrated Eclipse-based plug-in for facilitating the engineering of context-aware software services has not been addressed in existing work.

Table 12.2 Current study's context characteristics [4]

Scope of context												Representation of Context		Acquisition of context					Access of context	
Property (C.P.)								Extensibility (C.E.)	Chronology (C.C.)		Validity (C.V.)	Reusability (C.R.)	Abstraction (C.Ab.)	Automation (C.Au.)			Dynamicity (C.D.)		Mechanism (C.M.)	
Location	Temperature	Pressure	Time	Device	Network	User	Application		History	Future				Manual	Semi-automatic	Automatic	Static	Dynamic	Push	Pull
+	+	+	*	*	*	*	*	+	*	*	*	+	+	*	*	+	*	+	+	*

+ Explicitly supported; * not explicitly supported; − not applicable

Table 12.3 Current study's adaptation characteristics [4]

Kind of adaptation											Subject of adaptation								Process of adaptation						
Operation (A.O.)	Extensibility (A.Ex.)	Effect (A.Ef.)			Complexity (A.C.)		Level (A.L.)				Element (A.El.)						Granularity (A.G.)		Tasks (A.T.)	Automation (A.A.)			Dynamicity (A.D.)		Incrementality (A.I.)
		Add	Remove	Transform	Simple	Complex	Content	Hyperbase	Presentation	Others	Text	Audio	Image	Video	Link	Others	Micro	Macro		Automatic	Semi-automatic	Manual	Static	Dynamic	
+	+	+	*	+	*	+	–	–	–	+	–	–	–	–	–	+	+	*	+	+	*	*	*	+	+

Eclipse is a multi-language software development platform, which comprises an integrated development environment and a plug-in system to extend it. LTSA tool [23], which was originally created as a stand-alone tool, has now been extended to the Eclipse platform. LTSA Eclipse has an extensible architecture which allows extra features to be added by means of extended plug-ins. At present, the following plug-ins are supported which are available from the Eclipse install site: Message Sequence Chart, Architecture, WS-Engineer and SceneBeans. Similarly, the `Aspectual FSP Generation tool` can be integrated as an editor of the existing Eclipse-based LTSA tool. With this solution, service engineers can use a single integrated development environment to design and verify the pervasive software services specification for any property violations with much ease and confidence. At present, the current research performs the service engineering process using three stand-alone tools: LTSA-MSC tool, `Aspectual FSP Generation tool` and LTSA tool. In the integrated environment, the Message Sequence Chart plug-in can be used to specify the software services, generate and extract the core service model of the architecture while the `Aspectual FSP Generation plug-in` can be used to model the context-dependent adaptive behavior of the services using UML models and transform them into behavioral FSP. Finally, the LTSA can be used to perform model checking of the aspectual pervasive software services specification.

The extension of the `Aspectual FSP Generation tool` as an integrated plug-in to the Eclipse platform can be performed as follows. IBM Rational Software Architect [15], which is the development environment used to create the `Aspectual FSP Generation tool`, allows exporting of plug-in projects using an export wizard or a mechanism called the Plug-in Development Environment build. The Eclipse platform provides several notions to facilitate extension, which are plug-in, feature and update site [15]. A plug-in is the unit of new function contribution while a feature, which can include one or more plug-ins, is the unit of new function installation. An update site is a mechanism for finding and installing features. An update site can distribute one or more features. The multi-stage transformation chain developed in the current study includes several plug-ins, such as the back-end model-to-text JET transformation, the model-to-model mapping project, the `c-FSP-UML profile` project, and the EMF project. In order to export these plug-ins, first, a feature project that references those plug-ins needs to be created. Second, an update site needs to be created to distribute the feature created. The created update site can be deployed by copying the required files of the update site to a local or network folder, or to a Web server. Finally, the Eclipse Update Manager can be used to scan update sites for the newly created feature and install it.

12.8.2 Implementing the Model Checked Aspectual Pervasive Services

Service development is considered a very complex process that involves several stages of the software life-cycle, such as requirements analysis, design, implementation, testing and maintenance. In general, a service is validated during the testing phase, which is performed late in the software life-cycle. Testing the service code is considered costly, as any erroneous situations identified during the testing phase essentially require to reperform the design and implementation phases until the expected result is obtained. However, if service implementation can be generated automatically on an already-validated service specification using model transformation techniques, then it reduces or minimizes the need for testing the service code. This essentially reduces implementation time as the code is automatically generated, and the verified design and implementation levels of services are synchronized. Thus, reducing the need for any maintenance by the service engineer. The application of model transformations on an already-verified service design is also appropriate in the context of the current study.

This study has employed rigorous model checking to check whether individual aspects or components, and the woven model, contain any undesired behavior. This model checked pervasive software services specification, which is free of any erroneous behavior, can be fed into a custom model-to-code transformation tool created to automate the generation of executable service code or service implementation. Model-to-text transformations can be employed to generate both core and adaptable code of a service implementation. The core service code is the unchanging or static portion of the service while context handling or the adaptable code is the dynamic portion of the service, which can evolve based on available contextual information. In the current study, at the behavioral modeling level of FSP, the core service behavior and context-dependent information have been treated as separate concerns using the aspect-oriented modeling paradigm (`c-FSP aspects`). The same separation of concerns can be effectively enforced at the source code level with aspect-oriented programming. Model-to-text transformations can be employed to ensure the correct separation of concerns at both FSP and aspect-oriented programming levels. The service code can be provided using the aspect-oriented version of Java known as AspectJ of aspect-oriented programming and can target readily available software platforms such as Apache Tomcat Web Server and the Axis Simple Object Access Protocol engine. However, one of the main limitations of AspectJ is that it only supports compile time aspect weaving. In this regard, AspectWerkz can be a solution, which is a dynamic, lightweight and high-performing aspect-oriented programming framework for Java.

12.9 Conclusion

In summary, the primary contribution of this chapter is a novel, systematic architecture-centric approach for engineering context-aware services at the software architectural level. Context-awareness capabilities in service interfaces introduce additional challenges to the software engineer. The additional complexities associated with these special services necessitate the use of solid software engineering methodologies during their development and execution. To this end, this chapter has proposed a novel approach which integrates the benefits of solid software engineering principles of model-driven architecture, aspect-oriented modeling and formal model checking for engineering context-aware services. A prototype tool—`Aspectual FSP Generation`—applying an effective pipeline of model-to-model and model-to-text transformations has been built. The generated formal behavioral models for context-dependent behavior and the core service behavior have been rigorously verified using model checking against desired system properties. The approach has been explored using a real-world case study in intelligent transport, and an evaluation framework has been developed to validate the main methods and tools employed in the study. We have also discussed two key research directions, extending this work to benefit the broader service engineering and pervasive computing communities.

References

1. Abeywickrama, D.B.: Pervasive services engineering for SOAs. Ph.D. thesis, Faculty of IT, Clayton Campus, Monash University, Australia (2010)
2. Abeywickrama, D.B., Ramakrishnan, S.: A model-based approach for engineering pervasive services in SOAs. In: 5th International Conference on Pervasive Services (ICPS'08), Sorrento, Italy, pp. 57–60. ACM (2008)
3. Abeywickrama, D.B., Ramakrishnan, S.: Model-driven development of aspectual pervasive software services. In: 14th IEEE International Enterprise Distributed Object Computing Conference Workshops, Vitoria, Brazil, pp. 49–59. IEEE (2010)
4. Abeywickrama, D.B., Ramakrishnan, S.: An evaluation framework for validating aspectual pervasive software services. In: 6th International Conference on Evaluation of Novel Approaches to Software Engineering conference (ENASE'11), pp. 80–91. SciTePress (2011)
5. Abeywickrama, D.B., Ramakrishnan, S.: Model checking aspectual pervasive software services. In: 35th Annual IEEE International Computer Software and Applications Conference (COMPSAC'11), pp. 253–262. IEEE Computer Society (2011)
6. Abeywickrama, D.B., Ramakrishnan, S.: Context-aware services engineering: models, transformations, and verification. ACM Trans. Internet Technol. J. **11**(3), Article 10. ACM (2012)
7. Achilleos, A., Yang, K., Georgalas, N., Azmoodech, M.: Pervasive service creation using a model-driven petri net based approach. In: International Wireless Communications and Mobile Computing Conference, pp. 309–314 (2008)
8. Analyti, A., Theodorakis, M., Spyratos, N., Constantopoulos, P.: Contextualization as an independent abstraction mechanism for conceptual modeling. Inf. Syst. J. **32**(1), 24–60. Elsevier Science Ltd., Oxford, UK (2007)
9. Autili, M., Berardinelli, L., Cortellessa, V., Marco, A.D., Ruscio, D.D., Inverardi, P., Tivoli, M.: A development process for self-adapting service-oriented applications. In: International Conference on Service-Oriented Computing, LNCS, vol. 4749, pp. 442–448. Springer (2009)
10. Clarke, E.M., Wing, J.M., Alur, R.: Formal methods: state of the art and future directions. ACM Comput. Surv. **28**(4), 626–643. ACM (1996)
11. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge (1999)
12. Colombo, E., Mylopoulos, J., Spoletini, P.: Modeling and analyzing context-aware composition of services. In: International Conference on Service-Oriented Computing, LNCS, vol. 3826, pp. 198–213. Springer (2005)
13. Cottenier, T., van den Berg, A., Elrad, T.: Motorola WEAVR: aspect orientation and model-driven engineering. J. Object Technol. **6**(7), 51–88. Chair of Software Engineering, ETH Zurich, Switzerland (2007)
14. Davie, A.: Intelligent tagging for transport and logistics: the ParcelCall approach. Electron. Commun. Eng. J. **14**(3), 122–128. Institution of Electrical Engineers, London, UK (2002)

15. DeCarlo, J., Ackerman, L., Elder, P., Busch, C., Lopez-Mancisidor, A., Kimura, J., Balaji. R.S.: Strategic reuse with asset-based development. IBM Corporation (2008)
16. Dey, A.K., Abowd G.D.: Towards a better understanding of context and context-awareness. In: CHI 2000 Workshop on The What, Who, Where, When, Why and How of Context-Awareness (2000)
17. Douence, R., Botlan, D.L., Noye, J., Sudholt, M.: Concurrent aspects. In: 5th International Conference on Generative Programming and Component, Engineering, pp. 79–88 (2006)
18. Fuentes, L., Gamez, N., Sanchez, P.: Aspect-oriented executable UML models for context-aware pervasive applications. In: 2008 5th International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, pp. 34–43, Budapest. IEEE (2008)
19. Groher, I., Schulze, S.: Generating aspect code from UML models. In: 3rd International Workshop on Aspect-Oriented Modeling Co-located with 2nd International Conference on Aspect-Oriented Software Development (AOSD'03), Boston, USA (2003)
20. Hegering, H.-G., Küpper, A., Linnhoff-Popien, C., Reiser, H.: Management challenges of context-aware services in ubiquitous environments. In: Brunner, M., Keller, K. (eds.) Self-Managing Distributed Systems, LNCS, vol. 2867, pp. 321–339. Springer (2003)
21. Kapitsaki, G.M., Kateros, D.A., Prezerakos, G.N., Venieris, I.S.: Model-driven development of composite context-aware web applications. Inf. Softw. Technol. J. **51**(8), 1244–1260. Butterworth-Heinemann (2009)
22. Kappel, G., Pröll, B., Retschitzegger, W., Schwinger, W.: Customisation for ubiquitous web applications: a comparison of approaches. Int. J. Web Eng. Technol. **1**(1), 79–111. Inderscience Publishers, Geneva, Switzerland (2003)
23. Magee, J., Kramer, J.: Concurrency: State Models and Java Programs, 2nd edn. Wiley, New York (2006)
24. Mceachen, N., Alexander, R.T.: Distributing classes with woven concerns: an exploration of potential fault scenarios. In: 4th International Conference on Aspect-Oriented Software Development, pp. 192–200. ACM (2005)
25. Perez-Toledano, M.A., Navasa, A., Murillo, J.M., Canal, C.: TITAN: a framework for aspect-oriented system evolution. In: International Conference on Software, Engineering Advances, pp. 23–30 (2007)
26. Schwinger, W., Grün, C., Pröll, B., Retschitzegger, W., Schauerhuber, A.: Context-awareness in mobile tourism guides—a comprehensive survey.Technical report, Johannes Kepler University, Linz, Austria (2005)
27. Serral, E., Valderas, P., Pelechano, V.: Towards the model-driven development of context-aware pervasive systems. Pervasive Mobile Comput. J. **6**(2), 254–280. Elsevier (2010)
28. Sheng, Q. Z., Benatallah, B.: ContextUML: a UML-based modeling language for model-driven development of context-aware web services. In: International Conference on Mobile, Business, pp. 206–212 (2005)
29. Truong, H., Dustdar, S.: A survey on context-aware web service systems. Int. J. Web Inf. Syst. **5**(1), 5–31 (2009)
30. VIsualize all moDel drivEn programming (VIDE), WP 11: Deliverable number D11.3, Supported by the European Commission within Sixth Framework Programme. Polish-Japanese Institute of Information Technology. http://www.vide-ist.eu/download/VIDE_D11.3.pdf. Accessed 16 Sept 2012
31. Whittle, J., Jayaraman, P.: MATA: A tool for aspect-oriented modeling based on graph transformation. In: Giese, H. (ed.) Models in Software Engineering, LNCS, vol. 5002, pp. 16–27. Springer, Berlin(2008)
32. Xu, D., Alsmadi, I., Xu, W.: Model checking aspect-oriented design specification. In: 31st Annual IEEE International Computer Software and Applications Conference, pp. 491–500 (2007)