2011 35th IEEE Annual Computer Software and Applications Conference

Model Checking Aspectual Pervasive Software Services

Dhaminda B. Abeywickrama, Sita Ramakrishnan
Clayton School of Information Technology
Monash University, Clayton Campus
Victoria 3800, Australia
dhaminda.abeywickrama@gmail.com, sita.ramakrishnan@monash.edu

***Abstract*—Context-dependent information is tightly coupling or crosscutting the core functionality of a service at the service interface level. This results in a complex design, which is difficult to implement and maintain. The crosscutting context-dependent functionality of interacting pervasive services can be modeled as aspect-oriented models in UML. However, this has two challenges: the semi-formal nature of UML notations, and the expressive power of aspects. This paper explores model checking as a solution for *modeling* aspectual pervasive software services and their compositions, and rigorously *verifying* the process behavior of these models 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 pervasive aspects and components. These verification stages have been 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 with more than 30 properties formalized to provide a comprehensive coverage of the system requirements. An evaluation framework has been established to validate the main methods and tools employed in the study.**

***Keywords*-pervasive services; model checking; aspect-oriented modeling; model-driven development;**

I. Introduction

A pervasive Web service is a special type of service that adapts its behavior or the content it processes to the context of one or several parameters of a target entity in a transparent way (e.g. restaurant finder services, attractions and activities recommendation services, navigation and real-time traffic services, and dating services) [1]. With the proliferation of ubiquitous computing devices and the Internet, pervasive 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. Context information is characterized by several qualities that make pervasive 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, such as model-driven architecture, aspect-oriented modeling and formal model checking.

Context-dependent information is tightly coupling or crosscutting the core functionality of a service at service interface level. This results in a complex design, which is difficult to implement and maintain. The crosscutting context-dependent functionality of interacting pervasive services can be modeled as aspect-oriented models in UML. However, this has two challenges: the semi-formal nature of UML notations, and the expressive power of aspects. First, one of the main limitations of UML is its lack of support for rigorous verification due to its informal or semi-formal nature. Second, 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. In general, the crosscutting effect and oblivious principle of aspects can create several issues such as partial weaving, unknown aspect assumptions, unintended aspect effects, arbitrary aspect precedence, failure to preserve state invariants, and incorrect changes in control dependencies [2], [3]. Thus, the need for a more formal, rigorous specification and verification method for context-dependent adaptive behavior in service specifications is clear, such as model checking.

This paper explores model checking as a solution for *modeling* aspectual pervasive software services and their compositions, and *verifying* the process behavior of these models against specified system properties [4]. 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 pervasive 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 with more than 30 properties formalized to provide a comprehensive coverage of the system requirements. While several researchers [2], [3] have emphasized the challenges associated with the expressive power of aspects in design specifications, to the best of our knowledge there is no work that explores model checking as a solution to the crosscutting effects of *pervasive aspects* at

0730-3157/11 $26.00 © 2011 IEEE
DOI 10.1109/COMPSAC.2011.41

the service interface level. This verification approach is novel in this respect. An evaluation framework is established to validate the main methods and tools employed in the study.

The rest of the paper is organized as follows. Section II provides background information on the tools and techniques, the overall research methodology, the case study, and the context-dependent adaptive behavior generation process applied in this study. An overview of this authors' model checking process is provided in Section III. Sections IV and V discuss the aspect-oriented pervasive models created and concurrency modeling performed on the services. In Section VI the properties specification and verification of the models are elaborated. The evaluation framework established to validate the main methods and tools used is provided in Section VII. Section VIII summarizes related work, and Section IX concludes the paper.

II. Background

This section provides background information on (A) the tools and techniques, (B) the overall research methodology, (C) the case study, and (D) the adaptive behavior generation process applied in this study [4].

A. Tools and Techniques used

Model checking is an automatic technique for verifying finite state concurrent systems [5]. The Labeled Transition System Analyzer (LTSA) is a model checking tool for concurrency modeling, model animation and model property checking [6]. Finite State Processes (FSP) is a process calculus provided by the LTSA for concisely describing and reasoning about concurrent programs. In FSP, processes can be defined by using one or more auxiliary processes separated by commas and terminated by a full stop. Processes can be composed using the parallel composition operator and interactions can be modeled using shared actions, the renaming operator and the hiding operator. System properties can be defined using safety and progress property processes and fluent linear temporal logic (FLTL) assertions. The LTSA-MSC tool is an extension of the LTSA, which allows documenting scenarios in the form of graphical message sequence charts and generating a behavioral model in FSP.

B. Engineering Aspectual Pervasive Services

The overall pervasive service-oriented development process of this study is divided into three stages (Fig. 1) [4]. First, using the case study we extract use cases and define a service specification for the system under consideration using message sequence charts. Second, the architecture for the system is defined using a component configuration and an architecture model in FSP using the LTSA-MSC tool. Third, the architecture model synthesized from the previous step is modularized with aspect-oriented models in UML called the `contextual-FSP aspects` (`c-FSP aspects`), and automatically transformed into FSP before applying model checking using the LTSA tool.

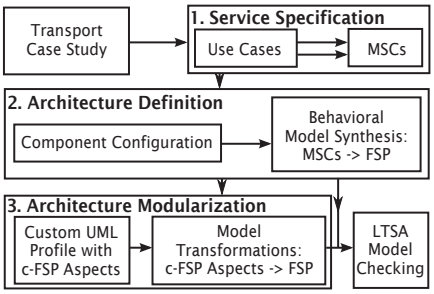


Figure 1: Pervasive services engineering.

C. Case Study: Awareness Monitoring and Notification

The research approach is explored using a real-world case study in intelligent tagging for transport known as the ParcelCall project [7]. ParcelCall [7] is 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 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.

D. Context-Dependent Adaptive Behavior Generation

The *notion of context* used in this research is based on a definition provided by Analyti *et al.* [8] for context in information modeling. They 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. Analyti *et al.* [8] use traditional object-oriented abstraction mechanisms of attribution, classification, generalization and encapsulation to structure the contents of a context.

The model transformation tool created in our study is called the `Aspectual FSP Generation tool` [4]. The transformations have been applied to the reference scenario in intelligent transport. 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. 2, the models and activities of the development process are represented as ellipses and square boxes respectively. The development process is structured into three main flows of activities. `Flow 1` and `Flow 2` extensively apply model transformations where `Flow 1` uses a model-to-text JET transformation and `Flow 2` applies an effective pipeline of model-to-model and model-to-text JET transformations. Both `Flow 1` and `Flow 2` originate from the `c-FSP-UML profile`. This profile describes our conceptual model to decouple crosscutting context-dependent information of a service from the core service behavior at service interface level. `Flow 3` represents activities involved for rigorously verifying the context-dependent adaptive behavior and the core service behavior of the pervasive software services using formal model checking, which is the focus of this paper.

III. Model Checking Aspectual Pervasive Software Services

In this section, we provide an overview of the activities involved in rigorously verifying the models generated for

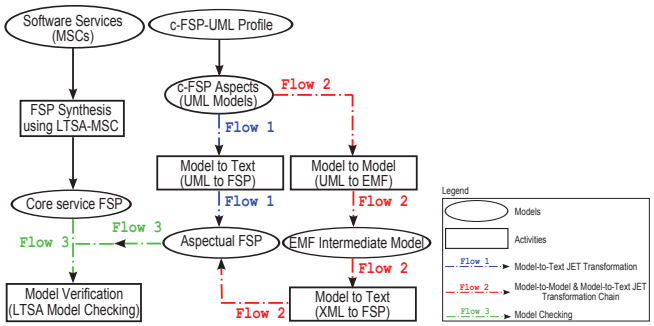


Figure 2: Adaptive behavior generation process.

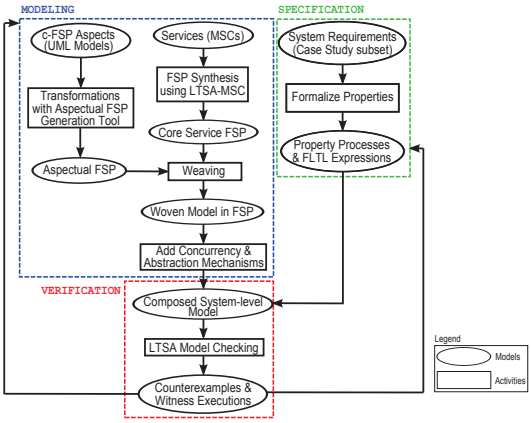


Figure 3: Model checking aspectual pervasive services.

the context-dependent adaptive behavior and the core service behavior using formal model checking [4]. This is shown by `Flow 3` in Fig. 2, and further elaborated in Fig. 3.

- *Modeling:* The modeling step involves two main tasks. They 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 (Fig. 2). 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 (Fig. 2, Fig. 3). 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 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:* The properties to be verified are formalized according to the system requirements, which are

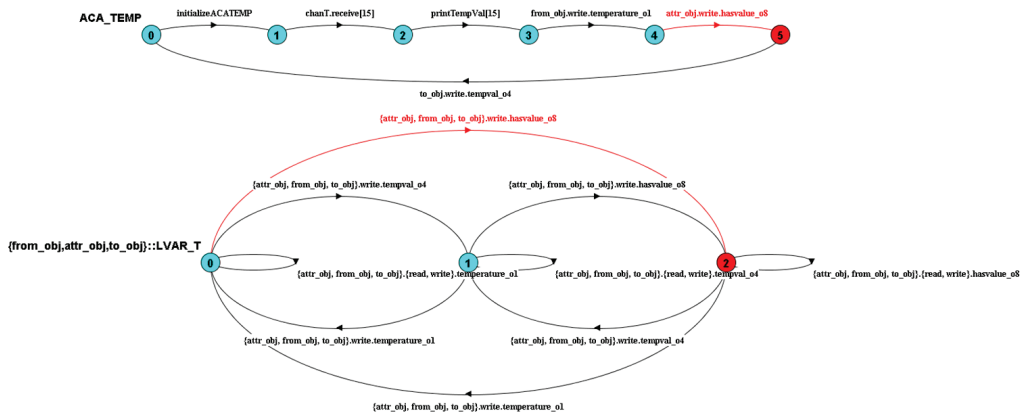


Figure 4: ACA_TEMP aspect and the label literals created for the attribution construct.

expressed as property processes (safety and progress) and FLTL assertions. According to the system requirements from the case study subset, more than 30 properties have been formalized focusing on the required behavior from both service components and aspects in the specification.

- *Verification:* Finally, all behavior and property processes are composed into a system-level process and this process is fed to the LTSA. The LTSA tool 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 model executions or 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 services.

IV. Pervasive Aspect-Oriented State Models

In this study an aspect is modeled as an independent state machine that synchronizes with the base state machine at specific synchronization points [4]. An aspect is defined as a modular encapsulation or unit for the crosscutting context-dependent behavior at the service interface level. The context procurement and contextualization activities of the `awareness monitoring and notification pervasive service` are driven by the `c-FSP aspects`. The current study identifies three types of aspects, which are collectively referred to as the `c-FSP aspects`. They are: `context aspects`, `trigger aspects` and `recovery aspects`.

There are two types of `context aspects`: `atomic context aspects` and `composite context aspects`. `Atomic context aspects` model low-level context readings from the context sources (e.g. `temperature`) while `composite context aspects` encapsulate high-level derived context information (e.g. `isAdverseStatus`). Also, the current research applies the notions of attribution, classification, generalization and encapsulation from the context definition [8] to structure and link the objects defined in a `context aspect`. The notion of context identifies that each object of a context can be a simple object or a link object (i.e. attribute, instance-of and ISA). Also, each object can be related to other objects through attribute, instance-of and ISA links. The notion of context defines three predicates to specify the link object, the source object and the destination object: $attr(attr\_obj, from, to)$, $in(in\_obj, from, to)$ and $isa(isa\_obj, from, to)$. The present study models these objects as *label literal variables* or *boolean variables* in FSP as required in the scenario. Variables in FSP may take an integer value or a label value. Label literals can be formed by preceding an action label with a quote (e.g. 'labelname). For example, the `Atomic Context Aspect Temperature` aspect (`ACA_TEMP`) contains three label literals for modeling $attr\_obj$, $from$ and to objects of the attribution construct. Furthermore, the process definition for this aspect includes transitions which read and write to these variables (Fig. 4). `Composite Context Aspect Adverse Environment Status` aspect includes an object to represent the high-level derived context information (`isAdverseState`). This object has been modeled using a boolean variable in FSP. The process definition for the aspect includes transitions for reading and writing from this boolean variable.

`Trigger aspect` (e.g. `Trigger Aspect Adverse Environment Status`) models the contextual adaptation where the service is automatically executed or modified based on context information. In general, a `trigger aspect` has a `context constraint` and an `action`. `Context constraint` of the `trigger aspect` is modeled as a conditional

transition or a guarded transition in FSP. `Action` of the `trigger aspect` is modeled as a transition in FSP, which instructs the `Notifier` component to send an SMS to the vehicle driver. `Recovery aspect` (e.g. `Recovery Aspect Adverse Environment Status`) models recovery actions that follow after an exceptional situation is raised by a `trigger aspect`. The execution of this aspect is dependent on the existence of the `trigger aspect`. For example, it may include transitions that control the refrigerator's temperature in the transport unit.

A. Joinpoint Model applied

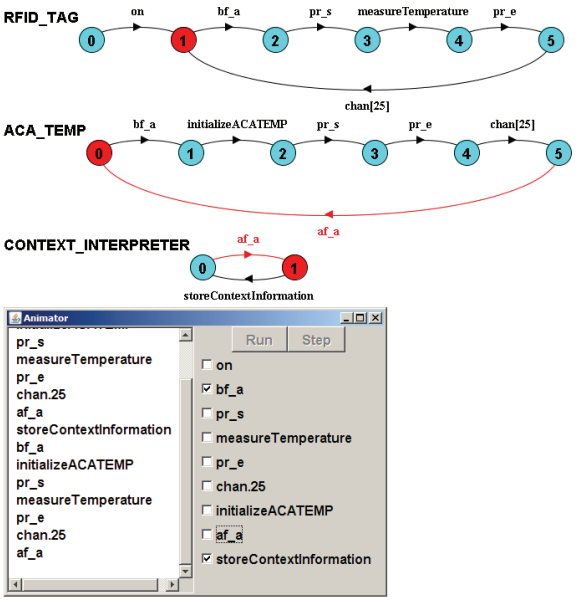
In this subsection, the *joinpoint model* applied in this research for the pervasive aspect-oriented state models in FSP is discussed. A main element in the design of any aspect-oriented language is the notion of the joinpoint model. The joinpoint model of our approach is used to facilitate the correct coordination of a base state machine and an aspectual state machine in FSP. Also, it is applied for weaving an aspectual state machine to its non-aspectual base state machine at specific synchronization points or joinpoints. This study concentrates only on dynamic crosscutting of the context-dependent adaptive behavior at the service interface level. Therefore, the specification of inter-type declarations has not been considered. The main crosscutting elements of the joinpoint model are:

- *Joinpoints or synchronization events:* A joinpoint is a transition in the base state machine where context-dependent adaptive behavior crosscuts the base state machine. It is effectively a transition in the base state machine which acts as a synchronization point with the aspectual state machine in FSP.
- *Transition pointcuts:* A pointcut in this study denotes a sequence of joinpoints.
- *Advice state models:* In general, an advice provides the actual crosscutting behavior which is defined in terms of pointcuts. In this study, an advice is specified using a finite state machine that describes the control logic or behavior applied to each joinpoint picked out by the pointcut. How an advice is executed depends on the type of advice used. This study models the *around advice* type in FSP, which comprises three parts: before actions, proceed actions and after actions. Proceed actions are compulsory control events while before and after actions can be optional.
- *Aspects state models:* In the study, an aspect can be identified as a modular encapsulation or unit for crosscutting context-dependent adaptive behavior at the service interface level. It effectively modifies the execution of the base state machine by adding new behavior. The notion of aspect applied here is *stateful*. An aspect state model contains an advice state model and pointcuts. The present research applies the notions of attribution, instance-of and ISA to formalize and modularize context information within an aspect. However, as our pointcut model defines a sequence of joinpoints as opposed to pairs of pointcuts and advice, in this respect this study's aspects state models can be considered much richer.

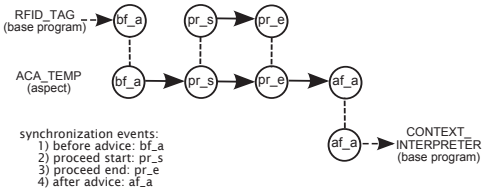
B. Pervasive Service Composition through Weaving

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. A base program (core service model) is specified as the parallel composition of the constituent base state machines in the model. In order to support explicit parallel composition, the current study injects synchronization events in both the aspectual state machine and the base state machine. These events provide an effective mechanism to control the coordination between an aspectual state machine and a base state machine. 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 followed are discussed next using a case study example. Fig. 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; thus, the sequence of `bf_a`, `pr_s`, `pr_e` and `af_a` constitutes the pointcut for this aspect. The execution and coordination of the base program and the aspect (Fig. 5b) can be explained as follows. The base program (`RFID_TAG`) emits the `bf_a` event to the aspect. The aspect performs an initialization operation (`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,



(a) Weaving illustrated using LTSs.



(b) Synchronization events.

Figure 5: Weaving performed.

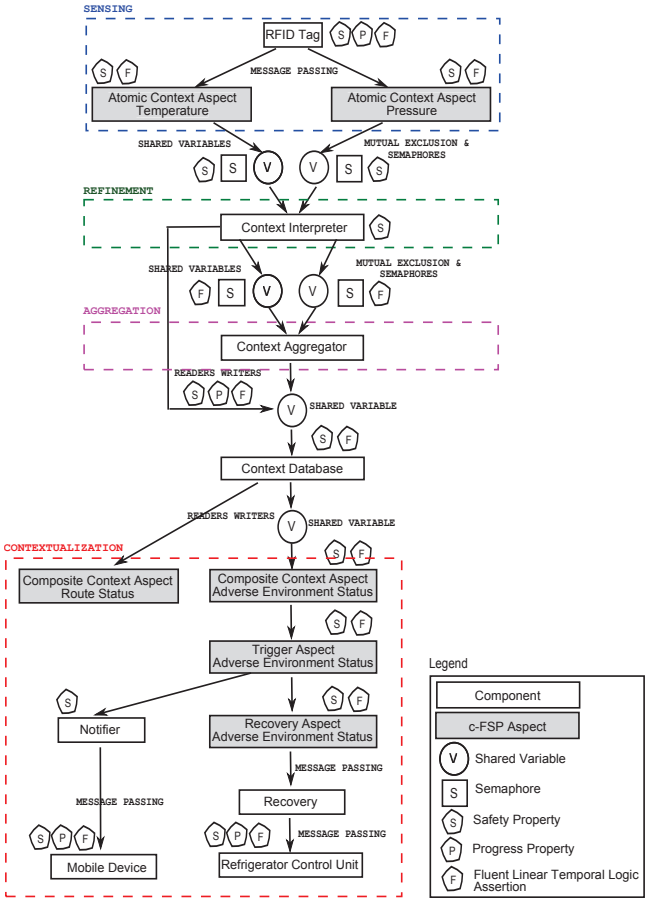


Figure 6: Concurrency modeling.

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.

V. Concurrency Modeling between Pervasive Aspects and Components

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 (Fig. 6) [4]. 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. Actions `a` and `b` are concurrent if they can occur in the order of `a -> b` or `b -> a`. Concurrency has been modeled using an interleaved model of concurrent execution.

This study models the `awareness monitoring and notification service` as a process-oriented context value chain. 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 is a mechanism 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.

Abstraction Mechanisms applied in the models: 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 the model checking technique is the state space explosion problem. In the present study, action hiding and minimization features available in FSP are used to reduce the size of the woven model before analyzing using the LTSA tool. For example, the actions or events 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 model checker.

VI. Properties Specification and Verification of Aspectual Pervasive Software Services

This section discusses the properties specification and verification stages of the model checking process [4].

A. Safety Requirements of the Study

Safety properties are used in a concurrent program to assert that nothing bad happens during the execution of the program. In the case study subset, several safety properties have been specified for verifying the behavior of the individual aspects and the components, and the behavior of a woven model. As for safety properties defined at the individual components and aspects level, a safety property (`S_TA_ADENST`) has been defined for the `Trigger Aspect Adverse Environment Status` aspect to verify whether a notification is sent only when environment status is adverse. A safety property (`S_CONTEXT_INTERPRETER`) has been defined for the `Context Interpreter` component to verify whether the refinement stage of the pervasive service is performed as expected.

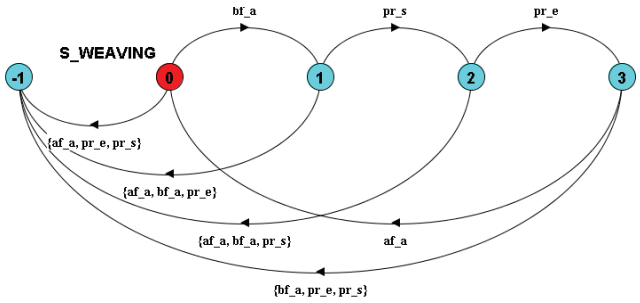


Figure 7: A safety property to verify weaving between base program and aspects.

In addition to performing safety analysis on individual components and aspects, this research performs safety analysis on a woven model level to verify its behavior. Safety properties have been defined to ensure the correct weaving of the base state machines and the aspectual state machines in the specification. For example, a safety property (`S_WEAVING`) has been defined to ensure the correct weaving between the following components and aspects in the specification: `RFID Tag`, `Atomic Context Aspect Temperature`, `Atomic Context Aspect Pressure` and `Context Interpreter` (Fig. 7). In this example, the `RFID Tag` and `Context Interpreter` components constitute the base program of the model. Synchronization events essentially represent the joinpoints where a base program is woven to the aspects. Synchronization events provide an effective mechanism to control the coordination between an aspectual state machine and a base state machine. `S_WEAVING` property ensures that the ordering of the synchronization events is correct in the components and aspects of the woven model, thus ensuring the correct weaving of the components and the aspects at the joinpoints in the specification. The `S_WEAVING` property is composed with the woven process before performing analysis using the LTSA. Analysis of this system using the LTSA shows that there are no deadlocks or safety violations.

Mutually exclusive access to shared variables between the components and the aspects have been modeled using semaphores in FSP. Safety properties can be created to verify whether the mutually exclusive access to the shared variables is enforced properly. For example, the `Atomic Context Aspect Temperature` aspect (`ACA_TEMP`) and the `Context Interpreter`

component (CONTEXT_INTERPRETER) processes share temperature values using the VAR_ATCI shared variable. In the same manner, the Atomic Context Aspect Pressure aspect (ACA_PRESS) and the CONTEXT_INTERPRETER component processes share pressure values using the VAR_APCI shared variable. Two mutual exclusion safety properties (S_AT_CI_MUTEX_T and S_AP_CI_MUTEX_P) have been created to ensure that when a process enters the critical section that process needs to exit before another process can enter it. Although LTSA analysis of this system shows that there are no deadlocks or safety violations, if the value of the semaphore is changed from one to two then the model produces a safety violation. This is clearly a violation of the mutual exclusion property as two processes have entered the critical section.

B. Progress Requirements of the Study

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 [6]. In the case study subset, progress properties have been specified for the readers-writers problem. To this end, two progress properties (P_READ, P_WRITE) 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. However, in order to find how the system performs when loaded, the action priority operator in FSP can be used to specify adverse scheduling conditions. In this example, this heavy loading has been modeled by making the release events lower priority. Analysis of this system using the LTSA reveals a P_READ progress violation (Fig. 8), which indicates that if a writer process is waiting to acquire the lock a reader process will never gain access to it. However, this P_READ progress violation can be disregarded as usually the number of write accesses to a database is less compared to the number of read accesses. Thus, the importance of preserving writer priority in acquiring the lock is clear as modeled in this example. The progress property P_REFRIG_CTRLUNIT ensures that the Refrigerator Control Unit component (REFRIG_CTRLUNIT) will eventually be enabled for it to receive any messages from the Recovery component using the synchronous message passing technique. A progress property (P_MOBILE_DEVICE) has been defined to ensure that the Mobile Device component will eventually receive reception for its correct operation.

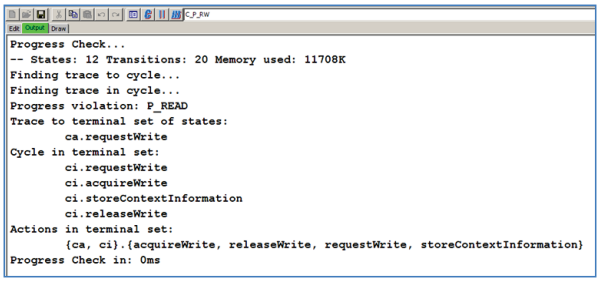


Figure 8: Counterexample for P_READ progress violation.

C. FLTL Requirements of the Study

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 [6]. The current study employs FLTL assertions as a method for specifying system requirements of the case study subset. Fluents provide several benefits over the property processes (i.e. safety and progress). First, fluents provide a more concise description of the required properties compared to property processes. Second, they express the required properties more directly compared to the property processes. Third, fluents facilitate the generation of witness executions to identify and locate potential errors in the specification. With property processes if a property is satisfied by the model, the LTSA does not return any further information. By contrast, FLTL properties provide the opportunity to generate traces or examples of model execution that satisfy the property which are referred to as witness executions.

Several FLTL properties have been defined for the case study subset. For example, two FLTL assertions (F_CI_CA_MUTEX_T, F_CI_CA_MUTEX_P) have been defined to ensure mutually exclusive access to the shared variables (valueTempCICA, valuePressCICA) by the Context Interpreter and the Context Aggregator service components (Fig. 9). 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. This research applies witness executions as a means of identifying potential errors in the specification. A FLTL property (F_WEAVING) has been defined to verify the weaving of the base state machines and aspectual state machines in the specification. The same property was defined as a safety property previously (S_WEAVING). The negation of the F_WEAVING assertion generates a counterexample, which was not possible with the S_WEAVING safety property process. By using counterexamples and witness executions, the state models and system properties for the

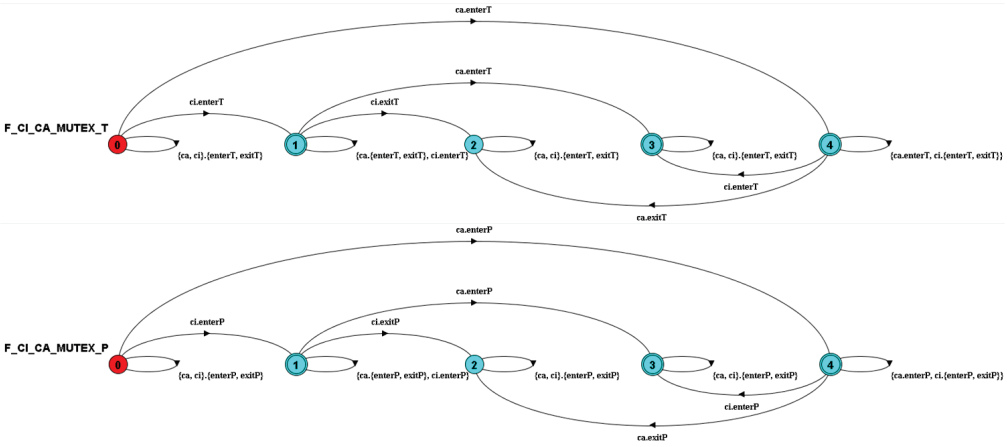


Figure 9: FLTL properties to ensure mutual exclusion.

aspectual pervasive services are iteratively improved.

VII. Evaluation Framework

We have established an *evaluation framework* to validate the main methods and tools employed in the study. The results are addressed in detail in [9]. The method of evaluation used here is based on key features comparison. The evaluation consists of a horizontal view and a vertical view (dimension) (Fig. 10). The horizontal evaluation view was designed to validate several desired key features required mainly at the platform-specific model (PSM) level (i.e. FSP) of the aspectual pervasive software services specification. These evaluation criteria mainly cover two aspects employed in the study. They are the formal methods and tools employed in the study, and the context and adaptation dimensions of the customization approach used in the pervasive services. In the vertical view, four research tools ([10], [11], [12], [13]) were compared to the `Aspectual FSP Generation tool` developed in this research. Like the `Aspectual FSP Generation tool`, these tools have been developed using commercially available toolchains of similar area of application. The tools were compared across the platform-independent model (PIM) and PSM levels of the model-driven architecture stack. This evaluation was based on several criteria: context-dependent behavioral modeling at the PIM level, explicit joinpoint model of aspect-oriented modeling at the PIM level, weaving performed at PIM or PSM levels, and context-dependent behavioral code generation from PIM to the PSM level.

The results of the evaluation are assuring. 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 indeed effective towards the overall objectives of this research. The vertical evaluation has demonstrated that the `Aspectual FSP Generation tool` has unique features in context-dependent behavioral modeling and context-dependent behavioral code generation. Like the `Aspectual FSP Generation tool`, [10], [12] and [13] 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.

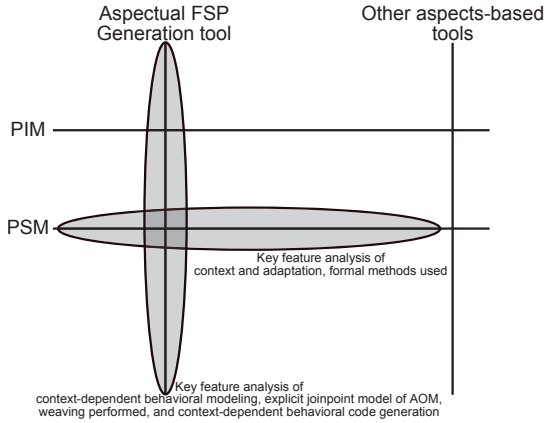


Figure 10: Evaluation framework: vertical, horizontal views.

VIII. Related Work

Douence *et al.* [14] propose a model for concurrent aspects which handles coordination issues between aspects and the base program and other aspects. Their approach is similar to our approach as both approaches use models of concurrently executing aspects and base state machines using FSP semantics of the LTSA. In [15], the authors present an approach to model checking state-based specification of aspect-oriented design. However, similar to [14], their approach is also not based on pervasive services. Furthermore, both these approaches do not use model transformations in their work as done in our research. A similar approach to ours, where pervasive services have been created using model-driven development is provided in [16]. However, in [16] validation of services is provided using petri nets. Also,

aspect-oriented modeling is not utilized in their approach for modularizing crosscutting context concerns.

From the analysis of related work, it is clear that there is very little work which applies model checking to verify context-dependent adaptive behavior at the service interface level. Also, to the best of our knowledge none of the approaches applies the software engineering techniques of model-driven architecture, aspect-oriented modeling and formal model checking, in the same approach. 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 pervasive services is novel, as performed here.

IX. Conclusion

This paper has explored model checking to address two challenges in context-dependent aspect-oriented UML models: the semi-formal nature of UML notations, and the expressive power of aspects. Model checking has been applied for modeling aspectual pervasive software services and their compositions, and rigorously verifying the process behavior of these models against specified 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. While several researchers have emphasized the challenges associated with the expressive power of aspects in design specifications, to the best of our knowledge there is no work that explores model checking as a solution to the crosscutting effects of pervasive aspects at the service interface level. This approach is novel in this respect. As for future work, the model checked pervasive software services specification, which is free of erroneous behavior, can be fed into a model-to-text transformation tool created to automate the generation of executable service implementation.

References

[1] H.-G. Hegering, A. Küpper, C. Linnhoff-Popien, and H. Reiser, "Management Challenges of Context-Aware Services in Ubiquitous Environments," in *Self-Managing Distributed Systems*, ser. Lecture Notes in Computer Science, vol. 2867. Springer Berlin / Heidelberg, 2003, pp. 321–339.

[2] N. McEachen and R. T. Alexander, "Distributing Classes with Woven Concerns: An Exploration of Potential Fault Scenarios," in *Proc. 4th International Conference on Aspect-Oriented Software Development (AOSD'05)*. Chicago, USA: ACM, Mar. 14–18, 2005, pp. 192–200.

[3] M. A. Perez-Toledano, A. Navasa, J. M. Murillo, and C. Canal, "TITAN: a Framework for Aspect Oriented System Evolution," in *Proc. 2007 International Conference on Software Engineering Advances (ICSEA'07)*. Cap Esterel, France: IEEE, Aug. 25–31, 2007, pp. 23–30.

[4] D. B. Abeywickrama, "Pervasive Services Engineering for SOAs," Ph.D. dissertation, Faculty of IT, Clayton Campus, Monash University, Australia, May 2010.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, UK: The MIT Press, Dec. 1999.

[6] J. Magee and J. Kramer, *Concurrency: State Models and Java Programs*, 2nd ed. John Wiley and Sons, Apr. 2006.

[7] A. Davie, "Intelligent Tagging for Transport and Logistics: The ParcelCall Approach," *Electronics & Communication Engineering Journal*, vol. 14, no. 3, pp. 122–128, Jun. 2002, Institution of Electrical Engineers, London, UK.

[8] A. Analyti, M. Theodorakis, N. Spyratos, and P. Constantopoulos, "Contextualization as an Independent Abstraction Mechanism for Conceptual Modeling," *Information Systems Journal*, vol. 32, no. 1, pp. 24–60, Mar. 2007, Elsevier.

[9] D. B. Abeywickrama and S. Ramakrishnan, "An Evaluation Framework for Validating Aspectual Pervasive Software Services (Accepted for Publication)," in *Proc. 6th International Conference on Evaluation of Novel Approaches to Software Engineering*, Beijing, China, Jun. 8–11, 2011.

[10] I. Groher and S. Schulze, "Generating Aspect Code from UML Models," in *Proc. 3rd International Workshop on Aspect-Oriented Modeling co-located with 2nd International Conference on Aspect-Oriented Software Development (AOSD'03)*, Boston, USA, Mar. 18, 2003.

[11] J. Whittle and P. Jayaraman, "MATA: A Tool for Aspect-Oriented Modeling based on Graph Transformation," in *Models in Software Engineering*, ser. Lecture Notes in Computer Science, vol. 5002. Springer, 2008, pp. 16–27.

[12] T. Cottenier, A. van den Berg, and T. Elrad, "Motorola WEAVR: Aspect Orientation and Model-Driven Engineering," *Journal of Object Technology*, vol. 6, no. 7, pp. 51–88, Aug. 2007, Chair of Software Engineering, ETH Zurich.

[13] L. Fuentes, N. Gamez, and P. Sanchez, "Aspect-Oriented Executable UML Models for Context-Aware Pervasive Applications," in *Proc. 2008 5th International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*. Budapest, Hungary: IEEE, Apr. 5, 2008, pp. 34–43.

[14] R. Douence, D. L. Botlan, J. Noye, and M. Sudholt, "Concurrent Aspects," in *Proc. 5th International Conference on Generative Programming and Component Engineering*. Portland, USA: ACM, Oct. 22-26, 2006, pp. 79–88.

[15] D. Xu, I. Alsmadi, and W. Xu, "Model Checking Aspect-Oriented Design Specification," in *Proc. 31st Annual International Computer Software and Applications Conference*. Beijing, China: IEEE, Jul. 23-27, 2007, pp. 491–500.

[16] A. Achilleos, K. Yang, N. Georgalas, and M. Azmoodech, "Pervasive Service Creation using a Model Driven Petri Net Based Approach," in *Proc. 2008 International Wireless Communications and Mobile Computing Conference*. Crete Island, Greece: IEEE, Aug. 6-8, 2008, pp. 309–314.