2011 35th IEEE Annual Computer Software and Applications Conference

An Approach to Model-based Development of Context-aware Adaptive Systems

Mahmoud Hussein, Jun Han, and Alan Colman
Faculty of Information and Communication Technologies, Swinburne University of Technology
PO Box 218, Hawthorn, 3122, Melbourne, Victoria, Australia
{mhussein, jhan, and acolman}@swin.edu.au

Abstract— **Many software systems need to be developed with adaptation in mind, where at runtime they need to detect changes in their operating environments and adapt themselves to cope with these changes while achieving or preserving the overall system goals. In recent years, there has been much research in the communities of context-aware systems and self-adaptive systems with their own foci. However, addressing context awareness and self-adaptation in a consistent and integrated manner remains a major challenge. In this paper, we introduce a novel approach to modeling and realizing such *context-aware adaptive software systems*. Our approach explicitly separates but relates the context model and the system model, so that their relationships, changes, and change impacts across the system and its contexts can be clearly captured and managed. In particular, we differentiate management context from operational context as to whether or not context changes cause system adaptation. To enable runtime changes to the system, its contexts, and their relationships, they all have their runtime representation, so that they can be manipulated and managed at runtime. Our component model for realizing the approach directly supports component interface definitions for context and management as well as functionality. Furthermore, we have developed a tool to generate the system implementations from their models and to validate and verify their context-aware adaptive behavior. We demonstrate our approach through the modeling and realization of a context-aware vehicle route planning system.**

Keywords - Context-awareness; Self-adaptation; System and Context Modeling; Runtime Models*; *Model Driven Development.

I. INTRODUCTION

There is an increasing demand for software systems that dynamically adapt their behavior at run-time in response to changes in their requirements, users' preferences, operational environments, and underlying infrastructure [1-2]. Changes can also be induced by failures or unavailability of parts of a software system itself [3]. In these circumstances, it is necessary for a software system to change itself as necessary to continue achieving or preserving its new and existing goals. A challenge is how to specify, design, verify, and realize such systems that evolve at runtime [1-5].

We focus in this paper on software systems that need to cope with the runtime context changes (i.e. changes in or to their operating environments), which we call context-aware adaptive software systems. Existing research into systems that adapt themselves in response to context changes has been carried out largely in two research communities with their own foci: self-adaptive systems [1-5] and context-aware systems [6-8]. On the one hand, research in context-aware systems is more concerned with how to represent, process, and manage the context information, but limited on how the system adapts itself in response to changes in the context information. On the other hand, research in self-adaptive systems is more about how to adapt the system in response to changes by separating the system functionality from its management and has paid less attention to how context is represented, processed, managed, and made available to the system. As a consequence, the system complexity is increased by hardwiring the context processing and management operations together with the system functionality and its management.

In this paper, we propose an approach that considers both the context-aware and self-adaptive perspectives in a holistic manner by capturing the system-context relationships explicitly over the stages of system modeling, realization and execution. Compared to existing work, our approach has the following novel features. Firstly, we explicitly *separate but relate* the system model and the context model, so that their relationships, changes, and change impacts across the system and its contexts can be clearly captured and managed. Secondly, in considering the system-context relationships, we differentiate management context from operational context. *Operational context* information is needed for the system to continue its normal operation while *management context* information causes the system to adapt itself. Thirdly, our approach enables the *runtime changes* to the system, its contexts and their relationships. We do so by maintaining runtime models for them, and these models are used in realising the changes while the system is in operation. Finally, we have introduced a component model for realising our approach that explicitly supports in the component interfaces the definition of the required and provided *functionalities, context information* and *management actions*.

A prototype tool has been developed to generate the context-aware adaptive software system implementations from their models. The tool also supports visual validation and formal verification of the systems' context-aware adaptive behaviour through a graphical user interface.

The remainder of the paper is organized as follows. We start by introducing a motivating scenario in section two. Our modeling and realization approach is given in section three. In section four, we demonstrate our approach and tool through the development of a context-aware adaptive route planning software system. Section five analyzes existing work with respect to our approach. Finally, we conclude the paper in section six.

II. RESEARCH MOTIVATION AND REQUIREMENTS ANALYSIS

The vehicle route planning software system helps the driver to plan his journey by providing suitable routes from the current location to a destination. Below are a few scenarios where this system takes into account different sources of context information in dynamically planning the travel route.

The context information may include (a) the dynamic traffic information available from road side units or a traffic information service provider (through Wi-Fi, DSRC and/or 3G technologies), and (b) the driver preferences (such as shortest or fastest or most carbon-efficient route) from his mobile phone. However, vehicles come in different models and with different enabling technologies. For example, some vehicles do not have the ability to communicate with the driver's mobile to get his preferences. The traffic information may or may not be available depending on the communication technologies installed and the availability of the road side units or traffic information provider. *Therefore, the vehicle route planning system should use different variants (i.e. algorithms) to compute the possible routes based on the available context information.*

When the vehicle is running in high speed such as over 70 km/h, it is difficult for the driver to concentrate on both the road and the displayed route map. In this situation, therefore, the system should use voice instruction for the vehicle route to reduce driver distraction. *As such, the system should add the voice instruction component when the vehicle is moving in high speed and the driver is not using the voice instruction option.*

The above scenarios show a number of general requirements. First, the system needs to take into account the context information to operate effectively (i.e. *context-aware*). For example, the traffic information and the driver preferences affect the possible routes computed by the route planner algorithms (*i.e.* part of the *operational relationships* between the system and its contexts). Second, the system should adapt itself (i.e. *adaptive*) in response to context changes. For example, adding the voice instruction component when the vehicle is moving in high speed (i.e. part of the *management relationships* between the system and its contexts). Third, the system should have the ability to *change the context model*, because different vehicles have different context information available depending on their communication technologies and some of this information may become unavailable for periods of time. Furthermore, the system should have the ability to *change the system-context relationships* at runtime. If the vehicle loses the ability to acquire the traffic information (due to communication failure or information unavailability), for example, then both the context model concerning traffic information and the corresponding system-context relationships need not be maintained to reduce system overhead. Therefore, the modeling and realization of the route planning system needs to consider both the context-aware and self-adaptive perspectives.

III. The Approach

In this section, we introduce our approach to modeling and realizing context-aware adaptive software systems. We start with our meta-model followed by our **c**ontext-aware **a**daptive syste**m**s com**p**onent model (*CAMP*), and then the **c**ontext-aware **a**daptive **s**ystems developmen**t** tool (*CAST*). To address the requirements highlighted in the above scenarios, our approach is aimed at (a) capturing clearly the system model, the context model, and their relationships, and (b) enabling their runtime adaptation in response to the context changes.

A. A Meta-Model for Context-aware Adaptive Systems

Our meta-model for context-aware adaptive software systems is shown in Figure 1. It is both a design-time and runtime model to support both system modelling and runtime context and system change. It consists of three composites that correspond to the context model (i.e. *context*), the system model (i.e. *functional system*) and their management relationships (i.e. *change management*). We discuss below the architecture and elements of this meta-model.

The Architectural considerations in the meta-model: The meta-model is particularly designed to address the requirements identified above.

(1) *Separate the context model from the functional system model*. Our meta-model (see Figure 1) represents the context model and the system model as two separate but related composites. As a consequence, the two aspects (i.e. the functional system and its context) and their relationships can be clearly captured.

(2) *Capture the operational and management system-context relationships*. As shown in Figure 1, we represent the *operational* system-context relationships via the direct links between the context and the functional system composites (e.g. providing the traffic information to the route planner component). The *management* system-context relationships are captured by the links between the context and the management composites and between the management and functional system composites. The management composite decides the adaptation actions in response to the context changes received from the context or the functional system composites, and then these adaptation actions are used to adapt any of the three composites. For example, the availability of the traffic information from the context triggers an adaptation action in the management, which in turn selects a specific realisation for the route planner component.

(3) *Enable the runtime adaptation of the context model, the functional system, and their relationships.* To enable the runtime change of the aforementioned system elements in response to the context changes, our meta-model captures both the static and dynamic aspects of the system elements.

The static system aspect concerns the structural (definitional) elements of the context model and the functional system, including the *entities* in the context representation and the *components* and *connectors* in the functional system representation (see Figure 1). The dynamic system aspect concerns the execution of the system, including the system *states* and *execution path(s).* The system *states* define the possible situations that the system elements can be in during execution, including the component *state field* of the functional system and the entity *attribute* of the context model in Figure 1. Examples are the response time of a functional system component and the current location of the vehicle. The *execution path(s)* are a set of rules that guide the transition between the different states of the system elements, i.e. the *management composite* in Figure 1. Based on the rule *condition(s)* (e.g. vehicle speed is greater than 70 km/h), a set of *adaptation actions* are triggered. These *adaptation actions* are for adding, removing, replacing, or modifying the system elements (e.g. adding the voice instruction component).

In the meta-model, we separate the system element definition (i.e. the *runtime representation*) from its realization (i.e. the actual system execution code). For example, we have the functional system component *definition* and its *realization* in the functional system composite, the entity *attribute* definition and its realization (i.e. *context provider*) in the context model composite, and the *adaptation action* definition and its *realization* in the change management composite (see in Figure 1). Doing so enables the change or swap of the element realization at runtime without affecting the system's overall structure and its elements relationships that are maintained at the system runtime representation layer. Furthermore, the runtime representations (and their realizations) can be changed at runtime to make the system have the ability to cope with the context changes.

The elements of the meta-model: Let us examine the meta-model in detail as shown in Figure 1.

(1) *The Context Model* (the top composite in Figure 1): To enable runtime adaptation of the context model, we have a runtime representation of the context model elements that are separate from but bound to the context providers.

(1.a) *The runtime representation of the context model*: The context is the information about the *entities* that are relevant to the system operation and/or adaptation. In Figure 1, we define the context model as a set of *entities* (e.g. the traffic information) that can have multiple *attributes*. These attributes can be low level context (i.e. single attribute such as *vehicle location*), or high level context (i.e. multiple attributes with a condition on them such as the *dangerous driving situation* that is deduced when the vehicle speed is greater than 70 km/h and the driver uses the route planner without voice instructions).

The context composite is an entity itself, and as such the context can have hierarchal levels for easy modelling and context information access. In addition, the context composite supports changes to the context model itself by adding, removing, or replacing a context entity or a context attribute to reduce the monitoring overhead at runtime.

(1.b) *Context providers*: The context information is provided by a set of context providers that can be *hardware* sensors (e.g. the On-Board Diagnostic (OBD) system for providing the vehicle speed), or *software* sensors such as the software component that provides the driver preferences (e.g. shortest or quickest route). A benefit of separating the context representation from the context providers is to enable the runtime selection of the context providers based on the required *context quality level* (e.g. context information accuracy, freshness, etc.). For example, the time period that passed between sensing the traffic information and sending it to the vehicle route planning application may make the received traffic information different from the actual one. Therefore, the system should take into account the traffic information freshness in selecting its provider (e.g. selecting the provider that has the up-to-minute traffic information).

(2) *The Functional System* (the bottom composite in Figure 1)*:* We have a runtime representation of the functional system that the change management layer can acquire the running system state from, and manipulate to achieve the runtime adaptation. We again separate the functional system realization from its representation.

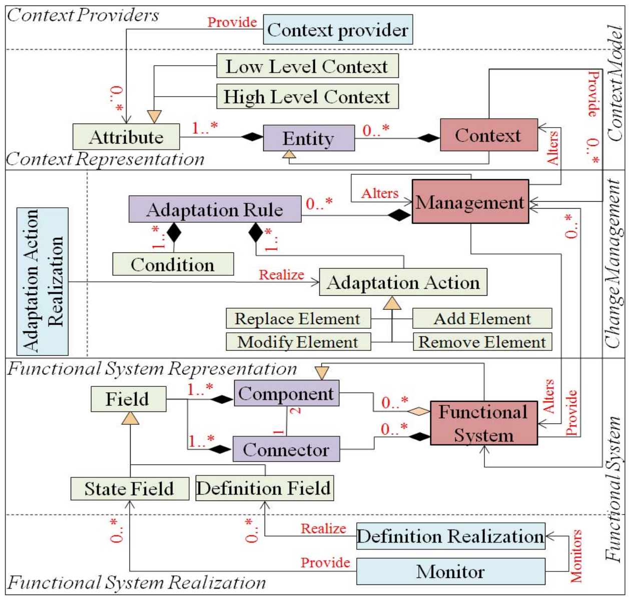


Figure 1. Meta-model for context-aware adaptive software systems

(2.a) *The runtime representation of the functional system*: The functional system consists of a set of *components* that provide the system's core functionality (e.g. the component that displays the vehicle routes), and *connectors* that capture the interactions between these components. The change management layer needs to know the current system *state* to take the suitable adaptation actions if needed. Therefore, each element in our functional system representation contains the *state* field to provide the current status of the component (e.g. the driver does not use the vehicle route voice instruction component) as well as the *definition* field for defining the system functionality (e.g. the vehicle route voice instruction function).

The functional system composite is a component itself and therefore supports system modelling at different levels of abstraction. In addition, it can be adapted by adding, removing, and/or replacing the system components and their connectors. Furthermore, the functional system composite is linked with the context composite for obtaining the required context information to continue its operation (i.e. the *operational system-context relationships*). For example, the functional system obtains the driver route preferences to be used by the route planning algorithm.

(2.b) *Functional system realizations*: The system components in the system representation layer are place holders for actual realization of component variations. For example, different realizations (algorithms) for the route planner can be available and one of them is used depending on the context information available. In addition, the system state information in the representation layer (e.g. the route planning algorithm response time) is provided by monitors in the functional system realization part.

(3) *The Changes Management* (the middle composite of Figure 1): This composite is responsible for capturing the *management system-context relationships* in the form of

adaptation rules. The context and/or the system state changes are used to trigger the evaluation of the adaptation rules conditions. When the conditions are evaluated to true, a set of adaptation actions are specified, and then they are performed on any of the three composites. The rule definitions are captured in the representation part, while their execution is by the adaptation action realization component.

An adaptation rule (as shown in Figure 1) is defined as having a set of *conditions* specifying when this rule is activated, and a set of *actions* defining the required adaptations when the rule conditions are evaluated to true. For example, when the traffic information becomes available in the context, the corresponding route planning algorithm in the functional system realization is selected.

In our meta-model, we use a set of general adaptation actions (add, remove, modify and replace) to simplify the diagram and they can be used for adapting any of the context model, the functional system, and the change management itself. However, they can be extended to include specific adaptation actions such as adding/removing context entity. Furthermore, the adaptation actions can have a system specific realization to ensure that they are performed without affecting the system consistency. For example, the component state can be transferred from a realization to another, in the case of switching between two different realizations for a system component.

We represent the adaptation rules as first class entities at runtime. Therefore, the change management itself can be adapted at runtime to improve the decision making performance by removing unwanted adaptive behaviours. For example, when the vehicle speed information is not available, its corresponding adaptation rules are disabled (to increase system efficiency).

B. *A Component Model for Context-aware Adaptive Systems*

To realize context-aware adaptive software systems using our meta-model, we introduce a **c**ontext-aware **a**daptive syste**m**s com**p**onent model (*CAMP*), i.e. all the aspects of the meta-model are realized through software components. As shown in Figure 2, the specialized component model is based on the concepts introduced in our meta-model, incorporating specific context and adaptation-related features into a traditional component model.

(1) *Required and provided function ports*: For representing the system functionality, our component model has the required and provided function ports. These ports are used for connecting the system components together. A component is connected with another if the former requires a function that is provided by the latter. For example, the *route planning display* component requires the computed routes that are provided by the *route planning algorithm* component and these two components are so connected. In addition, the provided function ports of multiple components can be connected with the required function port of another component to allow runtime selection, such as the selection of a route planning algorithm/component among the variants.

(2) *Required and provided context information ports*: The functional system needs the context information to *continue its operation*, e.g. the use of the traffic information in the route planning algorithm for computing the possible routes. Furthermore, the change management composite requires the system states from the system monitors to make adaptation decisions. Consequently, we explicitly reflect the requirement and provision of such context information in our component model through the required and provided context information ports.

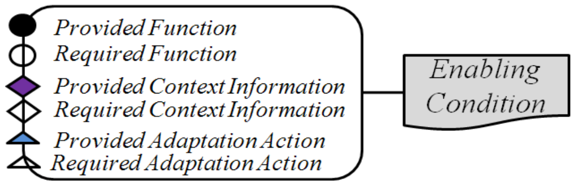

Provided Function
Required Function
Provided Context Information
Required Context Information
Provided Adaptation Action
Required Adaptation Action
Enabling Condition

Figure 2. Our context-aware adaptive systems component model (*CAMP*)

(3) *Required and provided adaptation action ports*: The change management composite is used to decide the required adaptation actions, and then its components (i.e. those related to adaptation rules) should explicitly define these required *adaptation actions*. In addition, actual adaptation actions need to be performed on the relevant components that should specify explicitly what adaptation actions they support. For example, the route planner component has the ability to switch between different route planning realizations. As such, our component model has explicit required and provided adaptation action ports as shown in Figure 2.

(4) *Enabling condition*: In our meta-model, we represent the system management as a set of adaptation rules, each of which in turn is as a set of conditions and actions. Therefore, we add the enabling condition element to our component model (see Figure 2) for the rule *condition definition* (e.g. vehicle speed is greater than 70 km/h).

In our component model, a component can be a simple component (i.e. Figure 2) or a composite component which consists of a group of components that are connected with each other. For example, Figure 3 shows a composite component that is composed of two simple components that are connected with each other via required and provided ports. In addition, the two required and provided ports of the simple components are exposed as required and provided ports of the composite component itself. This hierarchical view (i.e. *component composition*) enables system modeling at different levels of abstraction details.

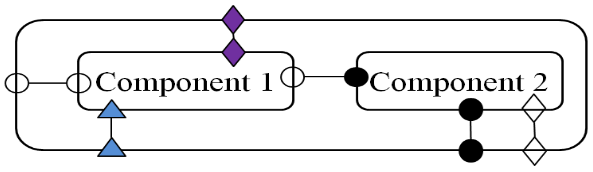


Figure 3. Our composite component model

Following the meta-model shown in Figure 1, an instantiated model that can be used by the software engineer to develop a context-aware adaptive software system using our component model is shown in Figure 4. It maps the concepts in the meta-model to our component model.

Firstly, the context providers and the functional system monitor components are represented as components that have only the provided context ports (e.g. *Context Provider1, and Monitor1*). Secondly, the context model, the functional system and the change management composites are represented as three separate composite components.

Thirdly, the context composite consists of sub-components that correspond to the meta-model entity concept (e.g. *Entity2*). Each sub-component has a set of provided context ports which correspond to the attribute concept in our meta-model (e.g. *Attribute1*). Furthermore, the context composite has provided adaptation actions to enable the context model adaptation (e.g. *Remove Entity2*).

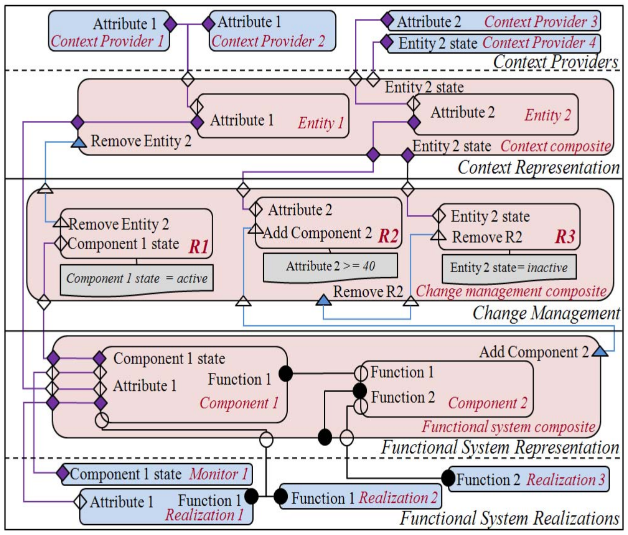


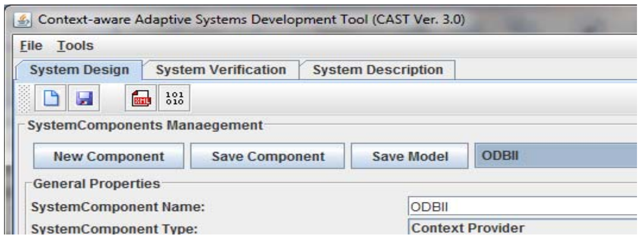
Figure 4. Mapping the meta-model concepts to our component model

Fourthly, the functional system composite consists of sub-components that provide its functionality, where each component defines what the provided and required functionalities are (i.e. *the definition field* in our meta-model). This composite also defines its *required context* ports (e.g. *Attribute1*) for linking with the provided context ports in the context composite (i.e. the operational relationships), *provided context* ports for providing functional system status such as response time (i.e. *state field*), and *provided adaptation action* ports that can be performed at the instruction of the change management composite (e.g. *Add Component2*).

Finally in Figure 4, the change management composite consists of sub-components that represent the system adaptation rules (i.e. the *management system-context relationships*). Each sub-component (e.g. R2) has the rule enabling condition(s) (e.g. *Attribute2 > 40*), and the rule action(s) as a required adaptation action port (e.g. *Add Component2*). In addition, the context attributes and system states in the rule conditions are exposed as required context ports of the rule component to obtain their values from the context or functional system composites. Furthermore, this composite also has some provided adaptation action ports to enable its own adaptation (e.g. *Remove R2*).

C. *Development Tool for Context-aware Adaptive Systems*

To support the development of context-aware adaptive software systems using our approach, we have developed a tool based on our meta-model and component model, the **c**ontext-aware **a**daptive **s**ystems developmen**t** tool (*CAST*). Its main screen is shown in Figure 5. It enables the software engineer to *model* and *generate* the system implementation, and to *validate* and *verify* the system's context-aware adaptive behaviour [9-11].



SystemComponent Name: ODBII
SystemComponent Type: Context Provider

Figure 5. Our context-aware adaptive systems development tool (CAST)

The following are the steps that the software engineer should follow when developing a context-aware adaptive software system using our approach and tool:

Step 1: *Create* a system model using our tool based on the component model and meta-model described above. Then a corresponding XML file can be generated automatically, capturing the system's model and following the XML schema shown in Listing 1. This schema describes the system as having at least three components (i.e. the functional system, the change management, and the context model), and each component has required and provided functionality (i.e. *ReqFunction* and *ProFunction*), context (i.e. *ReqContext* and *ProContext*), and/or adaptation actions (i.e. *ReqAction* and *ProAction*). In addition, each component's required ports define its connections with other components' provided ports (e.g. *FunctionProvider*). Each component can have sub-components if it is a composite component (i.e. *SubComponent*) or a condition if it is a rule component in the change management composite (i.e. *Condition*). A condition is an expression made of system states/context attributes (e.g. vehicle speed), arithmetic operators (e.g. greaterThan), attribute/state value (e.g. 70 km/h), and logical operators if any (e.g. and, or).

Step 2: *Verify* the system adaptive behavior. Using our tool, first, the system adaptive behavior model is transformed to a Petri Net. Second, the properties that need to be checked (e.g. adaptation rules redundancy, circularity, etc.) are generated in the form of temporal logic. Finally, a model checker is used to check the Petri Net model against these properties. More details about this formal verification step can be found in [11].

Step 3: *Generate* the system implementation. A set of Java classes corresponding to the system model is generated automatically. The generated classes for the context providers, the functional system realizations and the adaptation action realizations need to be completed by the system developer as their code is system specific.

Step 4: *Validate* the context-aware adaptive behaviour of the system visually. The system and the context providers are visualized and the software engineer can put specific context values. Then he can press the "*Adapt to the Context Information Changes"* button to see the context changes effect into the system. This feature enables the detection of *missing* and *incorrect* adaptation behaviours visually as will be shown in the next section. These two errors are not detected by the formal verification automatically, where they are related the system's adaptation requirements.

Step 5: After completing the system adaptive behavior test. If new adaptive behaviors are defined or the defined behaviors are changed, steps 1 through 4 are repeated again. When the software engineer is satisfied with the model, he passes the generated implementations to the system developer to complete the required code.

Listing 1: Context-aware adaptive system XML schema

```
<xsd:complexType name="System">
        <xsd:sequence>
          <xsd:element name="Component" type="Component Des" minOccurs="3"/>
        </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Component Des">
      <xsd:sequence>
        <xsd:element name="Name" type="xsd:string"/>
        <xsd:element name="ReqFunction" type="ReqFunction Des" minOccurs="0"/>
        <xsd:element name="ProFunction" type="xsd:string" minOccurs="0"/>
        <xsd:element name="ReqContext" type="ReqContext Des" minOccurs="0"/>
        <xsd:element name="ProContext" type="xsd:string" minOccurs="0"/>
        <xsd:element name="ReqAction" type="ReqAction Des" minOccurs="0"/>
        <xsd:element name="ProAction" type="xsd:string" minOccurs="0"/>
        <xsd:element name="Condition" type="Condition Des" minOccurs="0"/>
        <xsd:element name="SubComponent"   type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ReqFunction Des">
        <xsd:sequence>
          <xsd:element name="Name" type="xsd:string"/>
          <xsd:element name="FunctionProvider" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ReqContext Des">
        <xsd:sequence>
          <xsd:element name="Name" type="xsd:string"/>
          <xsd:element name="ContextProvider" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ReqAction Des">
        <xsd:sequence>
          <xsd:element name="Name" type="xsd:string"/>
          <xsd:element name="ActionProvider" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Condition Des">
        <xsd:sequence>
          <xsd:element name="Attribute" type="xsd:string"/>
          <xsd:element name="ArithmeticOperator"   type="xsd:string"/>
          <xsd:element name="Value" type="xsd:string"/>
          <xsd:element name="LogicalOperator"   type="xsd:string"/>
        </xsd:sequence>
</xsd:complexType>
```

IV. CASE STUDY

In this section, we follow the steps discussed above to develop the context-aware adaptive vehicle route planning software system as described in section two.

A. *The System Model*

We first create a model for the vehicle route planning software system as shown in Figure 6 (*Step 1*). This model has the following elements.

Context providers: In Figure 6, the context providers are: (a) the On-Board Diagnostic (OBD) system for providing the *vehicle speed* and its *availability* information; (b) the driver's mobile for providing his *route preferences*; (c) the traffic information service provider and road side units for providing the traffic *congestion information*.

Context representation: The context model has three entities (components): the vehicle information, the driver preferences, and the traffic information as shown in Figure 6. These entities represent the environment information that is needed by the route planning system to continue its operation or for triggering the system adaptation. In addition, the context composite is able to add, remove, or replace the context entities (e.g. *remove the traffic information* entity).

Functional system representation: It represents the system functionality and has two components as shown in Figure 6. First, the *route planner* provides the possible routes between the current location and destination. These routes are computed by different algorithms based on the available context information. The route planner component has the ability to switch among these different route planning algorithms implementations. For example, route planning two is used when the traffic information and the driver preferences are both available. Second, the *route planning display* presents to the driver the route computed by the route planner onto a map together with the journey progress and voice instructions. There are two variants for this component: (a) only the *map* with journey progress information over it (i.e. only the map component is used); (b) the *map* with the journey progress and *voice instructions* for the selected route (i.e. both the map and the voice instruction components are used). This variation is achieved by *adding and removing* the voice instruction component.

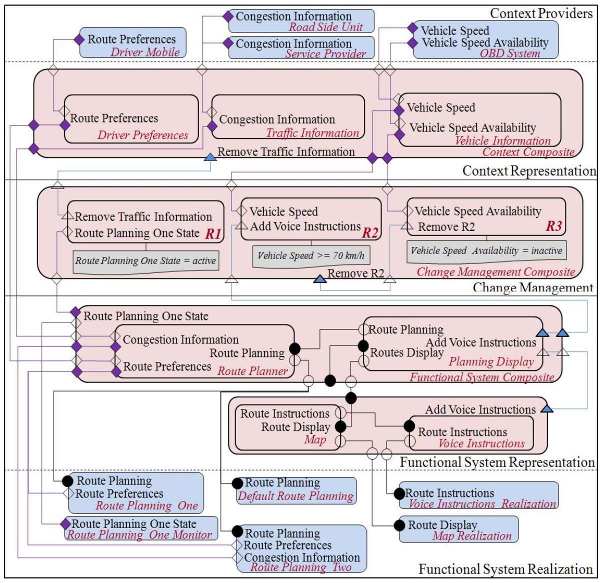


Figure 6. Context-aware adaptive vehicle routing planning example[1]

Functional system realization: In Figure 6, there are three different algorithms (i.e. *realizations*) for the route planner. The *default route planning* component takes the vehicle current location and the destination and provides the possible routes without taking into account any context information. The *route planning one* component considers the driver route preferences in calculating the routes. In addition, its state (i.e. the component is *selected and used* by the system or not) is provided by route planning one *monitor*. The component *route planning two* provides the available routes based on both the traffic congestion information and the driver route preferences. Besides, there are realizations for displaying the computed route onto a map and for providing the voice instructions for the selected vehicle route.

[1] To save the space, we modeled the system as a one level of abstraction, where our case study model is manageable. But, in a large scale system, the composite component can be used to model the system at different levels of abstraction details. As such, the system model visual display can be manageable.

Change management: This composite component consists of a set of rules that are used to determine the required adaptation actions in response to the context changes. Our example has many adaptation rules. To save space, we show only three adaptation rules that represent different adaptation categories (the middle part of Figure 6). These rules are for adapting the *context model* (R1), the *functional system* (R2), and the *change management* (R3).

(1) When the driver uses route planning one, the system needs to consider the driver route preference only, and then the traffic information context entity needs to be disabled to reduce the monitoring overhead. As such, component rule one in Figure 6 has the *enabling condition* "is the driver uses route planning one (i.e. *active*)?" and the *required adaptation action* "remove traffic information" context entity.

(2) In Figure 6, we define R2 as a component that evaluates to true when the vehicle speed is greater than 70 km/h (i.e. the *rule enabling condition*), and has the adding (i.e. *enabling*) of the voice instruction component as the *required adaptation action* to reduce the driver distraction.

(3) When the vehicle speed is not available, the evaluation of rule two is unwanted, and it adds overhead to the system and as such this rule should be removed or disabled. Therefore, the component R3 defines the unavailability of the vehicle speed as *its enabling condition* and has removal of R2 as the *required adaptation* action.

B. *The XML Description*

Following the XML schema of Listing 1, a part of the generated XML file for the context-aware adaptive route planning system model (i.e. Figure 6) is given in Listing 2. The XML file is structured as follows. Firstly, the *context model* is described as a set of components that represent the context entities. In Listing 2, the vehicle information context *entity* is a component, which has the vehicle speed as a context attribute. This attribute is provided via another component called the *OBD system* (i.e. a context provider).

Secondly, the *functional system* is described by a set of components that provide the system functionality and they are connected with each other via required and provided function ports. For example, the *Route Instructions* required port in the *Voice Instructions* component is connected with the *Route Instructions* provided port in the *Voice Instructions Realization* as shown in the middle part of Listing 2.

Finally, the *change management* is made of a set of components that represent the system adaptation rules. For example in Listing 2, component R2 describes a rule that has a condition, "*if vehicle speed is greater than 70 km/h"*, and the "*add voice instruction"* adaptation action. The complete XML description of the system model can be found in [10].

C. *Verfying the System Adaptive Behavior Formally*

The system adaptive behaviour may have errors such as inconsistency, redundancy, circularity, and incompleteness. First, the adaptation actions that are fired concurrently may contradict each other (i.e. *inconsistent*). For example, the fired adaptation actions are to add and remove the same system component. Second, if the same adaptation action is fired twice in the same context situation, then there is a *redundancy* in the system adaptive behaviour. Third, a chain of adaptation actions are fired continuously means that there is a *cycle* within the adaptation rules. Finally, if there is a context situation that has no reaction from the system, then the system adaptive behaviour is *incomplete*. As such, the system adaptive behaviour needs to be verified to detect these errors (*Step 2*).

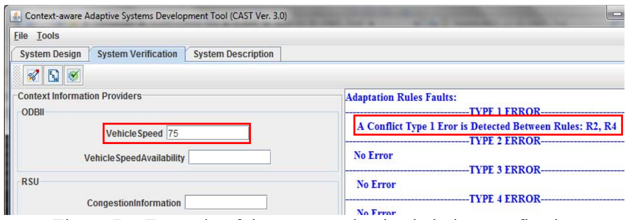
Listing 2: Route planning system XML description

```
<!-- Context Model Description -->
…
<Component>
     <Name> VehicleInformation </Name>
     <ProContext> VehicleSpeed </ProContext>
     <ReqContext>
          <Name> VehicleSpeed <Name>
          <ContextProvider> ODBSystem <ContextProvider>
     </ ReqContext >
</Component>
…
<! -- Functional System Description -->
…
<Component>
      <Name> VoiceInstructions </Name>
      <ProFunction> RouteInstructions </ProFunction>
      <ReqFunction>
          <Name> RouteInstructions <Name>
          <FunctionProvider> VoiceInstructionsRealization  <FunctionProvider>
       </ReqFunction>
</Component>
…
<!-- Change Management Description -->
…
<Component>
      <Name> R2 </Name>
      <ReqContext>
         <Name> VehicleSpeed </Name>
      </ReqContext>
      < ReqAction >
         <Name> Add_ VoiceInstructions </Name>
      </ ReqAction >
      <Condition>
         <Attribute> VehicleSpeed </Attribute>
        <ArithmeticOperator> GreaterThan </ArithmeticOperator>
        <Value> 70 </Value>
        <LogicalOperator> null </LogicalOperator>
      </Condition>
</Component>
…
```

For enabling the system adaptive behaviour verification, we linked our CAST tool with the Romeo tool [12]. This link is performed through generating two input files to the Romeo tool. First, a Petri Net corresponds to the system adaptive behaviour model is generated as an XML file. Second, a file is generated which has the properties (in the form of Timed Computational Tree Logic) that need to be checked (e.g. is the system adaptive behaviour consistent). Then, the Romeo tool is used to verify the system adaptive behaviour. After that, we obtain the verification results and display them in our tool in a user friendly manner.

We enable the specification of the Petri Net initial marking using our tool through a GUI that visualizes the context providers (the left part of Figure 7). Using this GUI, the software engineer specifies the context values (e.g. the vehicle speed is 75 km/h). These values are used for evaluating the adaptation conditions. When, a condition is evaluated to true (e.g. vehicle speed is greater than 70 km/h), its corresponding input place in the Petri Net is activated.

The adaptation rules in Figure 6 are simple, where we had simplified the system model, and then they do not have errors except missing adaptive behaviours. To show a type of error, we added a rule that remove the voice instruction component when the vehicle speed is lower than 80 km/h (R4). As such, when the vehicle speed is equal to 75 (the left part of Figure 7), the adaptation rules two and four are evaluated to true at the same time. As a consequence, the adaptation actions in this case are adding and removing the voice instruction component (i.e. *inconsistent* adaptation actions as shown in the right part of Figure 7). Details of our approach to verifying the system adaptive behavior and a complete case study can be found in [11].



Context-aware Adaptive Systems Development Tool (CAST Ver. 3.0)

File Tools

System Design | System Verification | System Description

Context Information Providers

ODBII

VehicleSpeed: 75

VehicleSpeedAvailability:

RSU

CongestionInformation:

Adaptation Rules Faults:

---TYPE 1 ERROR---

A Conflict Type 1 Eror is Detected Between Rules: R2, R4

---TYPE 2 ERROR---

No Error

---TYPE 3 ERROR---

No Error

---TYPE 4 ERROR---

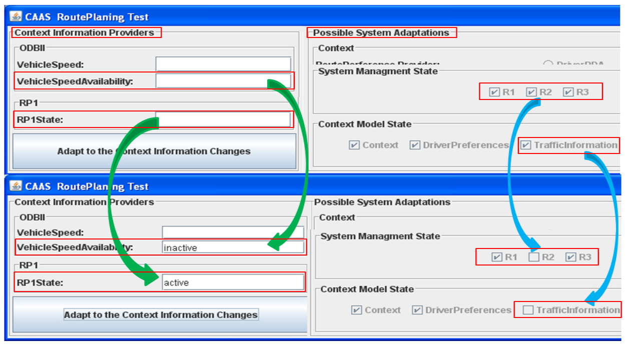
No Error

Figure 7. Example of the system adaptive behviour verfication

D. *Validating the System Adaptive Behavior Visually*

After the system model is created and its adaptive behaviour is formally verified (i.e. *Steps 1 and 2*), the system implementation is generated (i.e. *Step 3*). After that, the system adaptive behaviour is visually validated with regard to the system adaptation requirements by choosing *"Run the System Adaptive Behaviour Test"* from the tool's menu (i.e. *Step 4*). To do so, we generate a code that makes an instance of the system implementation and a GUI that is linked with this instance. This GUI visualizes the context providers, the context model, the functional system and the change management. Using this GUI, the software engineer can change the context situation by providing specific context values in the displayed textboxes. Then, by pressing the "*Adapt to the Context Information Changes"* button, the system implementation instance is adapted to the context changes and its state is displayed in the GUI.

Figure 8 shows an example, where the software engineer changes the vehicle speed availability value to be "*inactive*" and the route planning one state to be "*active*". This context situation activates the adaptation rules one and three: (a) the context model is changed by removing the traffic information context entity and (b) the change management is adapted by disabling the adaptation rule two. By repeating this process, (a) *missing* adaptation rules can be detected, if providing a context situation has no reaction from the system; (b) *incorrect* adaptation rules can be detected, if context changes lead to unexpected system reactions. After the detection and correction of the system's adaptive behaviour errors and the system engineer is satisfied with the design models, the generated implementation is completed by the system developers (i.e. *Step 5*). Details of this case and other case studies can be found in [9-11].



CAAS RoutePlaning Test

Context Information Providers

ODBII

VehicleSpeed:

VehicleSpeedAvailability:

RP1

RP1State:

Adapt to the Context Information Changes

Possible System Adaptations

Context

System Managment State

☑ R1 ☑ R2 ☑ R3

Context Model State

☑ Context ☑ DriverPreferences ☑ TrafficInformation

CAAS RoutePlaning Test

Context Information Providers

ODBII

VehicleSpeed:

VehicleSpeedAvailability: inactive

RP1

RP1State: active

Adapt to the Context Information Changes

Possible System Adaptations

Context

System Managment State

☑ R1 ☐ R2 ☑ R3

Context Model State

☑ Context ☑ DriverPreferences ☐ TrafficInformation

Figure 8. Example of the system adaptive behviour test

V. RELATED WORK AND DISCUSSIONS

A number of approaches have been proposed to support the development of context-aware adaptive software systems from self-adaptive and context-aware perspectives. We have performed an analytical survey of the existing research from both of these perspectives [13]. In this section, we briefly analyze the existing and our approaches in relation to the requirements we have identified.

Separation of Concerns: Existing approaches follow one of the two ways in system modeling. First, some separate the system functionality from its management but consider the context representation implicitly, i.e. no separate context model, as found in *self-adaptive systems research* [1]. As a consequence, the system modeling, realization and evolution become complex and error prone. Second, other approaches have an explicit context representation but hard-code the system management with its functionality, as found in *context-aware systems research* [6]. As such, they limit the system's runtime adaptation capability. In our approach, we separate the three aspects and keep them integrated from modeling to implementation and to runtime execution by capturing the system-context relationships explicitly. *As such, we can clearly capture and manage the system model, the context model, and their relationships.*

Runtime changes of the context model: The *context model* needs to be changed at runtime to *reduce the monitoring overhead* by only selecting the context model elements that are needed by the functional system. Most of the existing approaches have only a design time context model [7], and even those approaches that have a runtime context representation do not provide a method for managing the context model, i.e. they left this task to the system [14-16]. Therefore, the complexity of the (functional) system is increased, and the runtime context model changes become difficult. For example, in the MUSIC project [16], the context model elements are represented at runtime and when an element is needed it is activated. But, they do not provide a method of managing the context model, which they left that as the system's responsibility. *Our approach has a runtime representation of the context model and its management enables its runtime changes.*

Two types of contexts: There are two types of context information that needs to be considered: (1) the *environment context*, which is the environment information that affects the system operation; (2) the *system context*, which is the system states that the system management needs to know to initiate the adaptation process if needed. Current research considers either the environment context [15, 17-20], or the system context [21-23]. *Our approach takes into account both the environment and the system contexts explicitly.*

System-context relationships: The system-context relationships can be classified into *operational relationships*, and *management relationships*. Most of existing approaches consider these relationships implicitly [21, 23-24]. As such, the complexity of the system increases and the modeling and change of system becomes error prone. Even the approaches that consider these relationships explicitly [15, 20] consider only one type of these relationships (see Table 1). *In our approach, we represent the two types of relationships explicitly and separately.*

TABLE I. COMPARISON BETWEEN CURRENT CONTEXT-AWARE ADAPTIVE SYSTEMS RELATIVE TO OUR APPROACH

Requirements	Approaches	Rainbow [21]	Music [22]	Ayed et al. [25]	André et al. [26]	Andrade et al. [27]	CAPucine [28]	Scatter [29]	Heaven et al [23]	Zhang et al [30]	Morin et al. [24]	StarMX [31]	PLASTIC [32]	Folch et al. [33]	Oreizy et al. [34]	CAWAR [19]	CA3M [14]	ContextUML [15]	Serral et al. [20]	ScudWare [35]	Our Approach
Context Model	Low and high level context	+	+	~	+	~	~	~	~	~	~	~	~	~	~	+	+	+	+	+	+
	Runtime Representation	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	-	-	+
	Runtime Adaptation	-	~	-	-	-	-	-	-	-	-	-	-	-	-	-	~	-	-	-	+
Functional System	Structure/Behavior Adaptation	S	S	S/B	S	S	S	S	S	B	S	S	S/B	S	S	S	B	B	B	B	S
	System Context	+	+	-	-	-	+	-	+	-	+	-	-	+	+	-	-	-	-	-	+
	Runtime Adaptation	+	+	+	+	+	~	+	+	+	+	+	-	+	+	+	-	-	-	-	+
System-Context Relationships	Considered Context Type	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	O	O	O	O	M/O
	Explicit Relationships	-	-	+	-	-	-	+	+	+	-	-	+	-	-	+	-	+	+	-	+
	Relationships Realization	-	-	-	-	-	+	-	-	+	-	+	-	-	-	-	-	+	+	-	+
	Relationships Validation	-	-	-	-	-	-	-	-	~	-	-	-	-	-	-	-	-	-	-	+
	Relationships Adaptation	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+

(–) Unsupported (~) Partially supported (+) Supported (M) Management context (O) Operational context

Relationships runtime changes: Existing approaches do not maintain a runtime representation of the system-context relationships, and do not take their runtime changes into consideration [13]. *Our approach maintains a runtime representation of these relationships to enable their runtime change based on the context and system changes.*

Relationships specification and validation: Existing approaches use state-based models [23, 30] or utility functions [22, 33] to capture the *management system-context relationships*. But the process of specifying and implementing the management relationships using their techniques is complex in case of having a large number of adaptation behaviors [13]. In addition, using the condition-action rules to express them is easy [24, 31], but needs to be validated. In our approach, we adopted the condition-action rules where they are *expressive*. In addition, our component model supports their visual representation, and our tool is used to *visually validate* and *formally verify* these relationships as shown in section four.

System realization: The existing systems' component models only define explicitly the required and provided functional interfaces (e.g. Acme [36], Darwin [37], and etc.) or further have the provided adaptation actions interfaces (e.g. Fractal [38], ArchJava [39], and etc). Our component model extends them to include explicitly the (a) required and provided *context information*, (b) *required adaptation actions*, and (c) *adaptation conditions* (i.e. the component enabling condition) that define the situations where the system needs to adapt itself. As such, it provides an easy way to represent the *operational system-context relationships* by linking the required and provided context ports of the system components, and the *management system-context relationships* as a set of components. These management components specify the required adaptation actions interfaces based on the context state. Then, the required adaptation actions are linked to the system components that have the provided adaptation actions.

Table 1 summaries the analysis above and highlight (as *shaded*) our contributions as discussed above, and more details about this comparison can be found in [10].

VI. CONLUSIONS AND FUTURE WORK

In this paper, we have proposed an approach to *modeling and realizing* context-aware adaptive software systems. We have considered the functional system model, the context model, and their relationships explicitly from system modeling to realization and to execution. In addition, we have proposed a component model (*CAMP*) that incorporates explicitly support for definition of the system's functionality, context and management. It makes the capturing of the system-context *operational and management* relationships easier. Furthermore, we have developed a prototype tool (*CAST*) for generating the system implementations from their models, and verifying and validating the system adaptive behavior. We have also demonstrated our approach through the modeling and the realization of a context-aware adaptive vehicle route planning software system.

Compared to existing approaches, our approach has the following key contributions. First, the separation of the context model from the system model enables that the system-context relationships be clearly captured and managed. Second, it further differentiates and represents the operational system-context relationships from management ones to enable appropriate support. Third, it supports the runtime adaptation of the context model and the system-context relationships to reflect the runtime changes to the system and its contexts and reduce system overhead. Fourth, the system modeling task is made easier using our component model that directly incorporates context and management support. Furthermore, the system implementations can be generated from their models and the system adaptive behavior can be formally verified and tested visually.

As future work, our approach can be enhanced in several directions. First, software systems are usually deployed in environments which are not totally anticipated at the system design time. While we have a runtime representation of the system aspects (including rule-based management) to be able to cope with unanticipated context changes, runtime system management strategies and decision-making techniques are required to fully realize this capability. Second, we have applied our approach to several case studies [9-11], and the results were promising. We will perform more validations to assess the applicability and practicality of our approach. Finally, our approach supports only the changes in the system's parameters and structures. We will further consider the system's behavior changes [30].

ACKNOWLEDGMENT

This research was partly supported by the Australia's Cooperative Research Centre for Advanced Automotive Technology (AutoCRC) *(www.autocrc.com)*.

REFERENCES

[1] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.,* vol. 4, pp. 1-42, 2009.

[2] B. H. Cheng*, et al.*, "Software Engineering for Self-Adaptive Systems: A Research Roadmap," in *Software Engineering for Self-Adaptive Systems*, ed: Springer-Verlag, 2009, pp. 1-26.

[3] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," *Future of Software Engineering, 2007. FOSE'07,* pp. 259-268, 2007.

[4] M. Huebscher and J. McCann, "A survey of autonomic computing—degrees, models, and applications," *ACM Computing Surveys (CSUR),* vol. 40, p. 7, 2008.

[5] S. Dobson*, et al.*, "Fulfilling the vision of autonomic computing," *Computer,* vol. 43, pp. 35-41, 2010.

[6] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *Int. J. Ad Hoc Ubiquitous Comput.,* vol. 2, pp. 263-277, 2007.

[7] C. Bettini*, et al.*, "A survey of context modelling and reasoning techniques," *Pervasive Mob. Comput.,* vol. 6, pp. 161-180, 2010.

[8] H. Truong and S. Dustdar, "A survey on context-aware web service systems," *International Journal of Web Information Systems,* vol. 5, pp. 5-31, 2009.

[9] *http://www.ict.swin.edu.au/personal/mhussein/CAST.htm*, 2010.

[10] M. Hussein, J. Han, and A. Colman, "Integrated Modeling of Context-Aware Adaptive Software Systems," *Technical Report #C3-516_02, Swinburne University of Technology,* 2010.

[11] M. Hussein, J. Han, and A. Colman, "Specifying and Verifying the Context-aware Adaptive Behavior of Software Systems," *Technical Report #C3-516_03, Swinburne University of Technology,* 2010.

[12] D. Lime*, et al.*, "Romeo: A parametric model-checker for petri nets with stopwatches," *Tools and Algorithms for the Construction and Analysis of Systems,* pp. 54-57, 2009.

[13] M. Hussein, J. Han, and A. Colman, "Context-Aware Adaptive Software Systems: A System-Context Relationships Oriented Survey," *Technical Report #C3-516_01, Swinburne University of Technology,* 2010.

[14] C. Taconet*, et al.*, "CA3M: A Runtime Model and a Middleware for Dynamic Context Management," presented at the Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems: Part I, Vilamoura, Portugal, 2009.

[15] Q. Z. Sheng*, et al.*, "ContextServ: A platform for rapid and flexible development of context-aware Web services," presented at the Proceedings of the 31st International Conference on Software Engineering, 2009.

[16] R. Reichle*, et al.*, "A Comprehensive Context Modeling Framework for Pervasive Computing Systems," in *Distributed Applications and Interoperable Systems*. vol. 5053, R. Meier and S. Terzis, Eds., ed: Springer Berlin / Heidelberg, 2008, pp. 281-295.

[17] K. Henricksen and J. Indulska, "A Software Engineering Framework for Context-Aware Pervasive Computing," presented at the Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04), 2004.

[18] T. Gu, H. K. Pung, and D. Q. Zhang, "A service-oriented middleware for building context-aware services," *J. Netw. Comput. Appl.,* vol. 28, pp. 1-18, 2005.

[19] E. Mohyeldin*, et al.*, "A generic framework for context aware and adaptation behaviour of reconfigurable systems," in *Personal, Indoor and Mobile Radio Communications, 2005. PIMRC 2005. IEEE 16th International Symposium on*, 2005, pp. 1957-1963 Vol. 3.

[20] E. Serral, P. Valderas, and V. Pelechano, "Towards the Model Driven Development of context-aware pervasive systems," *Pervasive and Mobile Computing,* vol. 6, pp. 254-280, 2010.

[21] D. Garlan*, et al.*, "Rainbow: architecture-based self-adaptation with reusable infrastructure," *Computer,* vol. 37, pp. 46-54, 2004.

[22] R. Rouvoy*, et al.*, "MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments," in *Software Engineering for Self-Adaptive Systems*. vol. 5525, B. Cheng*, et al.*, Eds., ed: Springer Berlin / Heidelberg, 2009, pp. 164-182.

[23] W. Heaven*, et al.*, "A Case Study in Goal-Driven Architectural Adaptation," in *Software Engineering for Self-Adaptive Systems*, ed: Springer-Verlag, 2009, pp. 109-127.

[24] B. Morin*, et al.*, "Taming Dynamically Adaptive Systems using models and aspects," presented at the Proceedings of the 31st International Conference on Software Engineering, 2009.

[25] D. Ayed and Y. Berbers, "UML profile for the design of a platform-independent context-aware applications," presented at the Proceedings of the 1st workshop on MOdel Driven Development for Middleware (MODDM '06), Melbourne, Australia, 2006.

[26] Andre*, et al.*, "Towards a Generic Context-Aware Framework for Self-Adaptation of Service-Oriented Architectures," in *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on*, 2010, pp. 309-314.

[27] S. S. Andrade and R. J. de Araujo Macedo, "A non-intrusive component-based approach for deploying unanticipated self-management behaviour," in *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS '09. ICSE Workshop on*, 2009, pp. 152-161.

[28] C. Parra, X. Blanc, and L. Duchien, "Context awareness for dynamic service-oriented product lines," presented at the Proceedings of the 13th International Software Product Line Conference, San Francisco, California, 2009.

[29] J. White*, et al.*, "Automatically composing reusable software components for mobile devices," *Journal of the Brazilian Computer Society,* vol. 14, pp. 25-44, 2008.

[30] J. Zhang and B. H. C. Cheng, "Model-based development of dynamically adaptive software," presented at the Proceedings of the 28th international conference on Software engineering, Shanghai, China, 2006.

[31] R. Asadollahi, M. Salehie, and L. Tahvildari, "StarMX: A framework for developing self-managing Java-based systems," in *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS '09. ICSE Workshop on*, 2009, pp. 58-67.

[32] M. Autili, P. Benedetto, and P. Inverardi, "Context-Aware Adaptive Services: The PLASTIC Approach," presented at the Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering, ETAPS 2009, York, UK, 2009.

[33] J. Floch*, et al.*, "Using Architecture Models for Runtime Adaptability," *IEEE Softw.*, vol. 23, pp. 62-70, 2006.

[34] P. Oreizy*, et al.*, "An architecture-based approach to self-adaptive software," *Intelligent Systems and Their Applications, IEEE,* vol. 14, pp. 54-62, 2002.

[35] W. Zhaohui*, et al.*, "ScudWare: A Semantic and Adaptive Middleware Platform for Smart Vehicle Space," *Intelligent Transportation Systems, IEEE Transactions on,* vol. 8, pp. 121-132, 2007.

[36] D. Garlan, R. Monroe, and D. Wile, "Acme: an architecture description interchange language," presented at the Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research, Toronto, Ontario, Canada, 1997.

[37] J. Magee*, et al.*, "Specifying distributed software architectures," *Software Engineering—ESEC'95,* pp. 137-153, 1995.

[38] E. Bruneton*, et al.*, "The fractal component model and its support in java," *Software: Practice and Experience,* vol. 36, pp. 1257-1284, 2006.

[39] J. Aldrich, C. Chambers, and D. Notkin, "ArchJava: connecting software architecture to implementation," presented at the Proceedings of the 24th International Conference on Software Engineering, Orlando, Florida, 2002.