An Architecture-based Approach to Context-aware Adaptive Software Systems

Technical Report: C3-516_04

Mahmoud Hussein, Jun Han, and Alan Colman
Faculty of Information and Communication Technologies (FICT)
Swinburne University of Technology, Australia

January 2011
# Table of contents

**TABLE OF CONTENTS**

1. **INTRODUCTION**
2. **MOTIVATING SCENARIOS AND REQUIREMENTS ANALYSIS**
   2.1 Motivating Scenarios
   2.2 Context-aware Adaptive Software Systems Requirements
3. **CONTEXT-AWARE ADAPTIVE SOFTWARE SYSTEMS ARCHITECTURE**
   3.1 Context-aware Adaptive Software Systems Elements
   3.2 The Abstract Layered Architecture
   3.3 A Component Model for Context-aware Adaptive Systems
   3.4 Development Tool for Context-aware Adaptive Systems
4. **CASE STUDY**
   4.1 The Context-aware Adaptive Vehicle System Model
   4.2 Validating the Context-aware Adaptive Behavior Visually
   4.3 Verifying the Adaptive Behavior Using Romeo Tool
5. **RELATED WORK AND DISCUSSIONS**
   5.1 Context-aware and Self-adaptive Software Systems
   5.2 Context-awareness and Self-adaptivity Research
6. **CONCLUSIONS AND FUTURE WORK**
7. **REFERENCES**
Faculty of Information and Communication Technologies
Swinburne University of Technology

An Architecture-based Approach to Context-aware Adaptive Software Systems
Mahmoud Hussein, Jun Han, and Alan Colman

Abstract
Self-adaptive systems and context-aware systems have been proposed to provide the ability for a software system to adapt itself at runtime to cope with changes in its environment and user needs. However, research in self-adaptation and context-awareness has been carried out largely in separate communities, with limited reference to each other. Research in self-adaptation is more concerned with how to adapt the system, while research in context-awareness is more concerned with how to model, process, and manage the context information. In general, context-aware adaptive software systems need to consider both perspectives in a holistic manner. With the objective to gain a better understanding of the relationship between context-awareness and self-adaptation to advance the research and practice in this area, we in this paper introduce a layered architecture that integrates both aspects. In addition, we demonstrate our approach through the development of the context-aware adaptive vehicle route planning software system.

Keywords: Context-awareness, self-adaptation, context and system modelling, software architecture.

1 Introduction
There is an increasing demand for software systems that dynamically adapt their behavior at run-time in response to changes in their requirements, user 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 and/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].

Research into systems that adapt themselves in response to context and/or requirements changes has been conducted by researchers from two communities with two different emphases: 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 model, process, and manage the context information. But limited on how a system adapts itself in response to (unanticipated) changes in the context information. In addition, it is usually not concerned about requirements changes while the system is in operation. On the other hand, research in self-adaptive systems is more about how to adapt the system in response to context and/or requirements changes by separating the system functionality from its management and has paid less attention to how context is modeled, processed, managed, and made available to the system. In practice, the line between the two is rather blurred, and the engineering of a context-aware adaptive system needs to consider both aspects in a holistic manner. With the objective to gain a better understanding of the relationship between context-awareness and self-adaptivity to advance
the research and practice in this area, we in this paper introduce a layered architecture that integrates both aspects. To realize the approach, we used our component model that explicitly supports in the component interfaces the definition of the required and provided functionalities, context information and management actions [9]. In addition, our developed tool supports the generation of the context-aware adaptive software system implementations from their models [10]. The tool also supports visual validation and the formal verification of the systems’ context-aware adaptive behaviour through a graphical user interface [11]. Finally, we demonstrate our approach through the development of the context-aware adaptive vehicle route planning software system.

The remainder of the paper is organized as follows. We start by introducing a motivating scenario from context-aware adaptive vehicle systems in section two. In section three we present our approach. In section four, we demonstrate our approach through the development of a context-aware adaptive route planning system. Section five analyses existing work with respect to our approach. Finally, we conclude the paper in section six.

2 Motivating Scenarios and Requirements Analysis

The context-aware vehicle systems are examples of context-aware adaptive software systems, which need to adapt themselves according to changes in their environments, and the driver/passengers needs. We present below a number of scenarios where such vehicle systems are used. They will serve to (1) help the discussion of the report; (2) show the need to integrate context-awareness and self-adaptation aspects in developing this type of systems.

2.1 Motivating Scenarios

James’s vehicle is equipped with context-aware software applications to navigate and organize his affairs while on the road. The vehicle/applications’ connectivity with the outside world (including other vehicles, smart road infrastructure, services over the internet etc.) is enabled by built-in Bluetooth, DSRC, 3G and GPS technologies. The applications in his vehicle include the Vehicle-Navigation-Assistant (VNA), which assists him in route planning, congestion avoidance, and communicating with other vehicles, smart road infrastructure, and external commercial services that match his preferences such as services that provide the traffic information and the locations of nearby pharmacies. Below are a few scenes, he experiences when driving home one day.

**Scene 1**: The trip starts when James entering his vehicle. The vehicle detects James and activates his profile. The vehicle starts to play his favorite music, select his default route, and display a message, “you need to get the medication,” that has been specified in his diary beforehand.

**Scene 2**: James needs to re-plan his way home to stop at a pharmacy. The vehicle system turns on the VNA application to search for and communicate with the best available services that provide the pharmacies locations, and the traffic information while considering his preferences (e.g. lowest service price, good communication link quality, etc.). Then, the VNA shows the available routes to him based on the current context (James preferences, vehicle location, pharmacy location, and the traffic information), and then he chooses the suitable route and starts his journey home.

**Scene 3**: While he was driving, first, the traffic information service provider become unavailable, and then the VNA finds another service provider for the traffic information and communicates with it. Second, the road side units send the local area speed limit to the VNA, and then the vehicle speed limit is adjusted. Third, rain is detected by some dedicated sensors of his vehicle. He is informed of
the speed limit, and he changes the vehicle speed accordingly. In addition, James decided to use the
adaptive cruise control (ACC), which automatically adjust the vehicle speed to keep a specified
distance with the preceding vehicle while considering the rain level (i.e. the ACC operational behavior
in rainy conditions). When the rain stopped, the ACC is returned to its normal operational behavior.
Finally, congestion is happened en-route, and then the VNA shows to James the available routes, and
then he chooses the suitable route and continues his journey home.

2.2 Context-aware Adaptive Software Systems Requirements

The above scenario shows that the context-aware adaptive vehicle system needs to:

(1) Model, process, and manage its context information: First, the system’s environment has a large
amount of information about the driver, the vehicle, and the vehicle environment (e.g. the nearby
vehicles, the services providers, the road side units, etc.) which affects the system operation, and
then a model of the context information need to be maintained. Second, the raw context data is
required (e.g. the front vehicle speed and location), but higher level context information is more
powerful (e.g. the possibility of colliding with this vehicle) and needed by the system. As such, the raw
context data need to be processed to infer high level context information. Finally, during the system
operation the context information are changeable, where the context is not totally anticipated during
the system design time (e.g. the number of the nearby vehicle). Consequently, the system needs to
have a runtime representation of its context to enable its changes.

(2) Adapt itself in response to context and driver's needs changes: First, the system need to adapt
itself in response to context changes. For example, if the traffic information service provider becomes
unavailable, a new service provider needs to be selected and used. Second, in response to driver’s
needs changes, the system adapts itself to incorporate these changes. For example, the driver can
request the route planning and then the VNA is turned on, the change of the desired distance gap
with the preceding vehicle, and an application for vehicle diagnosis assistant (i.e. unanticipated
requirements change).

(3) Capture the relationship between the context changes and the system reactions: The context
information can be classified into two types based on their effect into the system. First, the operational
context is needed by the system to continue its operations. For example, to find the nearest pharmacy
location, the vehicle current location is needed. Second, the management context needs system to
adapt itself. For example, if the driver specified in his diary to get the medication, then the system’s
structure is changed by communicating with a service that provide the location of nearest pharmacy.

3 Context-aware Adaptive Software Systems Architecture

The above scenario show that the development of context-aware adaptive systems need to consider
the context-awareness perspective (i.e. model, process, and manage the context information), self-
adaptation perspective (i.e. the system adapt itself to keep achieving its goals), and their integration
(i.e. the relationships between context changes and the system reactions). In this section, we present
our approach that takes into account the above aspects.

3.1 Context-aware Adaptive Software Systems Elements

The context-aware adaptive software system elements that are needed to address the above three
requirements are shown in Figure 1. First, the context side represents the environment where the
system is working in and its status is captured using a context model as shown in the bottom of
Figure 1. Second, the system side is consisted of the functional system and its management. The functional system represents the system’s core functionality and a set of resources (i.e. the operational context) that is needed to achieve this functionality. The system management is responsible for deciding the required adaptation actions in response to context changes. Finally, a set of control loops are used to capture the relationship between the system and its contexts.

To enable the system response to changes in the functional system status, an internal control loop is formed between the functional system and its management, where the system management monitors the functional system and analyzes its status to detect the changes that needs the system to adapt. Then, the required adaptation actions are decided and acted into the system.

The environment context changes have different relationships with the system management. First, a feed-back control loop relationship is formed, when the system reaction (i.e. adaptation actions) changes the environment variable that initially causes the system adaptation. As such, when this variable is modified, its value should be fed again to the system management. For example, the system needs to maintain a distance gap with the preceding vehicle. To do so, the vehicle speed should be adjusted depending on the current distance gap, which in turn affects the distance gap. After that, the distance gap is measured again and fed to the system to adjust the vehicle speed if required. Second, the feed-forward control loop relationship is constructed when the context information is fed to the system and the system reactions have no effect into this context information. For example, when the rain level is fed to the system, the system adjusts the vehicle speed accordingly which has no effect into the rain level. Consequently, the system needs to have feed-back/feed-forward control loops to keep the system and its context consistent at runtime by reflecting the context changes into the system.

The functional system needs the context information to continue its operation. For example, the route planning algorithm needs the traffic information to calculate the possible routes effectively. As such, a relationship between the functional system and its context need to be represented.

The context-aware adaptive system has different relationships among the functional system, the system management, and the context model. These relationships shares similar operations such as monitoring/sensing, analyzing/processing, deciding, and acting. As such, they can have the same/similar realizations. In addition, the separation between (a) the system and its context, and (b) the functional system and its management need to be considered. Consequently, in Figure 2 we show the different kinds of relationships that are required. This Figure contains (1) context-functional system relationship, which represents the relationship between the functional system and the
operational context; (2) the control loop that starts from the sensors and ends at the effectors to capture the relationship between the system’s/environment context and the system management. To use this control loop in developing context-aware adaptive system, there is a need for architecture that links the system elements and their relationships which is described in the next sub-section.

3.2 The Abstract Layered Architecture

Our abstract layered architecture for context-aware adaptive software systems is shown in Figure 3, and it has three layers as follows:

The functional system and its context (layer one at the bottom of Figure 3): This layer has three elements. First, the functional system is consisted of a set of components that are used to achieve the system core functionality. During the runtime, these components can be running components (i.e. C1 and C2) or inactive components (i.e. operational context) that can be used by the functional system when needed (i.e. SC1, SC2, and SC3). Second, the context contains a set of environment entities that affect the system operations and/or adaptations (i.e. EC1 and EC2). Finally, the interfaces with the management layers (i.e. layers two and three). These interfaces are (a) a set of sensors/monitors to detect the changes in the system or its context, and then they inform the higher layers to take the required adaptation actions, and (b) a set of effectors to apply the adaptation actions that have been taken by the management layers into the running functional system.
The system and its context representation (layer two): This layer has a representation of the system and its context and has two operations. First, the monitoring/sensing operation is responsible for maintaining up-to-date (a) context model, where the environment context information are organized into a context model; (b) system models, which they are corresponding to the system state such as system’s structure and/or behaviour models. Second, the acting operation is used for applying the actions that are coming from layer three to (a) the running system models first to assure that there is no volition of the system state consistency (i.e. the system ends up with a valid state), and then to the running system; (b) the context model, where inferred high level context information using the processing operation at layer three are stored into the context model or the context model need to be evolved to cope with unanticipated context changes.

The change management (layer three): This layer maintains a precise specification of the system’s functional requirements (i.e. FR model) and the non-functional requirements such as performance, security, and etc. (i.e. NFR models) that need to be preserved at runtime. In addition, it has three operations. First, the analysing operation checks the consistency among the running system state, the environment state, and the system requirements, to initiate the deciding operation in case of consistency violation. Second, the processing operation is used for inferring the high level context information. Finally, the deciding operation specifies the required adaptation actions in response to context and/or requirements changes, and the current system’s configuration/behaviour is not designed to deal with this change. There are a set of pre-defined adaptation actions (i.e. adaptation script) in response to an expected class of state changes. However, when none of the available adaptation is suitable to cope with these changes, the deciding operation tries to generate new adaptation script on-the-fly. If the generation of new adaptation script is not possible, the system administrator is informed to perform the required changes (i.e. define manually the required changes).

Our three layers partition the system in a way that allows the context-aware adaptive systems requirements that are listed in section two to be handled using different layers. First, the context is modelled at layer two, processed and managed at layer three. Second, the change management layer is responsible for adapting the system to cope with context and/or requirement changes. In addition, we separate the system realization from its runtime representation (i.e. runtime models) to enable the system runtime adaptivity. Third, the monitoring and acting operations at layer two captures the system-operational environment context relationship, where the context is monitored and then passed to the system directly. In addition the whole control loop at layers two and three is used to capture the relationship between the context and the system management, where the system adapts itself in response to context changes.

3.3 A Component Model for Context-aware Adaptive Systems

To realize our context-aware adaptive software systems architecture, we introduce a context-aware adaptive systems component model (CAMP), i.e. all the aspects of the architecture are realized through software components. As shown in Figure 4, the specialized component model is based on the concepts introduced in our layered architecture, incorporating specific context and adaptation-related features into a traditional component model.

(1) Required and provided function ports: For representing the system functionality (i.e. the functional system), 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.

Figure 4: Our context-aware adaptive systems component model (CAMP)

(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 system management 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.

(3) **Required and provided adaptation action ports:** The system management is used to decide the required adaptation actions, and then its components 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 4.

(4) **Enabling condition:** 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 4) for enabling the rule condition definition.

In our component model, a component can be a simple component (i.e., Figure 4) or a composite component which consists of a group of components that are connected with each other. For example, Figure 5 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 5: Our composite component model

Following the architecture shown in Figure 3, 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 6. It maps the concepts in the architecture to our component model.

Firstly, the context and the functional system sensors/monitor are represented as components that have only the provided context ports (e.g., Context Provider1, and Monitor1). In addition, the system
effectors are represented as components that have provided adaptation actions only (e.g. Effector 1). Secondly, the context composite consists of sub-components that represent the environment entities (i.e. the context model). Each sub-component (e.g. Entity2) has a set of provided context ports which correspond to the entities attributes (e.g. Attribute2). In addition, the context composite has provided adaptation actions to enable the context model adaptation (e.g. Remove Entity2).

Thirdly, 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 functional system model). The system composite required functionalities are provided by a set of components (e.g. Realization 1) that have the provided functions ports only (i.e. the functional system). 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 system-context relationships), provided context ports for providing functional system status such as response time, and provided adaptation action ports (i.e. functional system effectors) that can be performed at the instruction of the system management composite (e.g. Add Component2).

Finally in Figure 6, the system management composite consists of sub-components that represent the system adaptation rules (i.e. the management system-context relationships, in other words the deciding operation). Each sub-component (e.g. R2) has the enabling rule condition(s) (e.g. Attribute2 > 40), and the rule action(s) as a required adaptation action port (e.g. Add Component2). 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 components. In addition, a simple analysis process can be defined via a set of conditions of a component. To define a situation that is inferred when a context attribute value is equal to V1 and another context attribute value is greater than V2, a component is defined. This component has this situation as provided context port and an enabling condition "contextAttribute1=V1 and contextAttribute2>V2". Furthermore, the system management composite also has some provided adaptation action ports to enable its own adaptation (e.g. Remove R2). This changeability feature enable incorporating new system reactions to context information introduced at runtime.

![Figure 6: Mapping the architecture concepts to our component model](image)

3.4 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 architecture and component model, the context-aware adaptive...
systems development tool (CAST) [10]. It enables the software engineer to model and generate the system implementations, and validate/verify its context-aware adaptive behaviour. 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 the architecture described above. Then, a corresponding XML file is generated automatically, capturing the system’s model and following the XML schema describe in [9].

**Step 2:** Generate the system implementation. A set of java classes corresponding to the system model (i.e. the generated XML file) is generated automatically. The generated classes for the context providers (i.e. the sensors and monitors), the functional system realizations and the adaptation action realizations (i.e. the effectors) need to be completed by the system developer as their code is system specific (i.e. the architecture layer one).

**Step 3:** Validate/verify the context-aware adaptive behaviour of the system [11]. The system and the context providers are visualized and the software engineer can put specific context values or select specific context model configuration. 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 the incorrect adaptation behaviours visually. In addition, the system adaptive behaviour can be verified to identify errors such as inconsistency, redundancy, etc. To do so, we transform the adaptive behaviour model to Petri Net [12] and the properties that need to be checked into temporal logic [13]. Then, Romeo tool is used to perform the verification [14].

**Step 4:** After completing the system adaptive behavior validation and verification. If new adaptive behaviors are defined or the defined behaviors are changed, steps 1, 2 and 3 are repeated again. When the software engineer is satisfied with the model, it passes the generated implementations to the system developer to complete the required code (i.e. the actual implementations of the context providers, the functional system realizations, and the adaptation actions realizations).

4 Case Study

In this section, we use the steps described above to develop the context aware-adaptive vehicle system that was described in section two with a particular focus on the vehicle route planning system. We present the system model in sub-section one, the system adaptive behaviour visual validation by our tool [10] and verification using Romeo tool [14] are discussed in sub-sections two and three.

4.1 The Context-aware Adaptive Vehicle System Model

The context-aware adaptive vehicle route planning system model that represents an instance of our abstract layered architecture described above is shown in Figure 7. In the following, we describe the architecture three layers and their elements to show the applicability of our approach.

Layer one (the system and its context): This layer is consisted of a set of components that are used to provide the system core functionalities and the functional system interfaces (i.e. the system’s monitors, sensors, and effectors) with the environment and the management layers (i.e. layers two and three). In addition, each system component has the ability to change itself if required (i.e. adding, removing, replacing function(s) at the component level).
In Figure 7, layer one contains the realizations that provide the system’s two basic functions: the route planning and the route display. First, the route planning component is realized through three different algorithms. 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. The component route planning two provides the available routes based on both the traffic congestion information and the driver route preferences. Second, there are realizations for displaying the computed route onto a map and for providing the voice instructions for the selected vehicle route for realizing the route display component.

There are a set of providers for the context information (i.e. the sensors and monitors in Figure 7). First, the On-Board Diagnostic (OBD) system provides the vehicle speed. Second, the driver’s mobile is used for providing his route preferences. Third, the traffic information service provider and road side units provide the traffic congestion information. Forth, the availability of the traffic information and the driver preference context are obtained through their entities. Finally, the route planning one state (i.e. the component is selected and used by the system or not) is provided by route planning one monitor.

In our case study, the system’s adaptation actions are generic, which do not need system’s specific change (e.g. components states transfer). As such, we do not have system’s effectors at layer one.
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, the route planning two is used when the traffic information and the driver preferences are both available. The route display component 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 using only the map component; (b) the map with the journey progress and voice instructions for the selected route using both the map and the voice instruction components. This variation is achieved by adding and removing the voice instruction component.

The acting operation is performed via the components that have provided adaptation action ports in their interfaces (e.g. remove voice instruction component in the route display component). In addition, the monitoring operation obtains the values from layer one sensors and monitors to keep the models up-to-date (e.g. updating the vehicle speed value in the vehicle information context entity).

Layer 3 (the change management): During the runtime, this layer is responsible for analysing the context information and then deciding the required adaptation actions to preserve the system’s high level goals. In our approach, we represent the system management (see Figure 7) as a set of rules that are used to determine the required adaptation actions in response to the context changes (i.e. the deciding operation). The analysis operation in our approach is simple, which it is performed via a set of conditions on the context values (not shown in Figure 7). Our example has many adaptation rules. We show only six adaptation rules. These rules are designed to have different adaptation behaviours types (e.g. context model, functional system and system management changes) and errors (e.g. redundancy, circuitry, etc.) for highlighting the capabilities of our approach.

(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 for reducing the monitoring overhead. As such, the component rule one (R1) in Figure 7 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) The traffic information providers can be disabled due to communication link problems during the vehicle journey, and then we defined the adaptation rule two (R2). This rule makes the system switches to using the route planning one (i.e. the required adaptation action), when the traffic information is not available (i.e. the rule enabling condition).

(3) The availability of the driver route preferences enables the selection of the route planning one. Therefore, the component rule three (R3 in Figure 7) defines the availability of route preference as the rule enabling condition and the use of route planning one as the required adaptation action.

(4) The route planning algorithm two is used when both the driver route preference and the traffic information are available. To represent this case, we define the adaptation rule four (R4) which have the availability of this context information as the condition to use the route planning algorithm two.

(5) In Figure 7, we define R5 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.
(6) When the driver is driving in low speed, the voice instruction may be annoying, and then it should be removed. In Figure 7, R6 evaluates to true when the vehicle speed is lower than 80 km/h, and has the removal of the voice instruction component as the required adaptation action.

4.2 Validating the Context-aware Adaptive Behavior Visually

The system adaptive behaviour can be visually validated by choosing “Run the System Adaptive Behaviour Test” from the tool’s menu in our CAST tool [10]. To enable this feature, we generate the system implementations from its model. Then, a code is generated that makes an instance of these implementations and a GUI that is linked with this instance. This GUI visualizes the context information providers, the context model, and the functional system. 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 into the GUI.

Figure 8 shows an example, where the software engineer changes the driver route preference availability value 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 functional system is adapted by selecting the route planning algorithm one. By repeating this process, incorrect adaptation behaviours can be detected, if context changes lead to unexpected system reactions (invalid system state).

![Figure 8: Testing the system adaptive behaviour visually example](image)

4.3 Verifying the Adaptive Behavior Using Romeo Tool

For enabling the adaptive behaviour verification using Romeo tool [14], we linked our tool with it. This link is performed through generating (a) a Petri Net that is corresponding to our adaptive behaviour model as an XML file and (b) the properties that need to be checked as TCTL (Timed Computational Tree Logic) file (i.e. the input files format of the Romeo tool). Then, we use the model checker
implemented inside the Romeo tool for checking the adaptive behaviour, and then we get the verification results and display them in our tool in a user friendly manner [11]. In addition, we enable the specification of the Petri Net initial marking using our tool through a GUI that visualized the context providers (Figure 9-A). The specified context values are used for evaluating the adaptation rules condition, and then the condition that is evaluated to true its input place in the Petri Net is activated (i.e. have a token). When the software engineer press the verification button, the Petri Net model is generated, the model checker is called, and then the verification result is display as in Figure 9-B.

For example, when the vehicle speed is equal to 75 (Figure 9-A), the adaptation rules five and six are evaluated to true in the same time. The adaptation actions in this case are adding and removing the voice instruction component, and then a conflict is detected between the rules evaluated to true (i.e. R5 and R6) as shown in Figure 9-B.

![Figure 9: Verifying the adaptive behaviour using Romeo tool](image)

By repeating the above process several times, we have detected the following errors in the specified adaptive behaviour:

**Error 1**: The adaptation rules five and six actions can be triggered simultaneously when the vehicle speed value is between 70 and 80 km/h. Therefore, a conflicting action can happen (i.e. add then remove the voice instruction component).

**Error 2**: The driver route preferences and the traffic information context entities can be active in the same time. As consequence, the adaptation rules three and four are triggered together which means there are two replacements for the route planning algorithm in the same context situation.

**Error 3**: Rule one is to change the context model in response to the functional system change (i.e. remove the traffic information when the route planning one is active). In addition, when the traffic information is not available (i.e. context model change), the route planning one is selected (i.e. functional system changes). These two rules have cycles, where the activation of one rule makes the other active and versa which leads to an infinite loop between them.

**Error 4**: When there is not any context information available, the system should use the default route planning. However, the specified adaptive behaviour does not have this rule (i.e. missing adaptive behaviour). In addition, there are other missing adaptation behaviours, where we only show a simplified example to highlight possible error and different system adaptations. The designed adaptive behaviour model is free from redundancy error, where there is no duplication in the adaptation rules and there is no adaptation rule that is a part of another.

### 5 Related Work and Discussions

The development of context-aware adaptive software systems needs to consider the context-awareness and self-adaptation perspectives in a holistic manner (see our scenario and its analysis in section 2). However, current research pays attention to either the self-adaptation or the context-awareness aspects. In the following section, we analyse existing approaches with regard to our
proposed layered architecture. Sub-section one describes what context-aware and self-adaptive systems are intending to do when they initially proposed. The state of existing research into context-awareness and self-adaptation is discussed in sub-section two.

5.1 Context-aware and Self-adaptive Software Systems

Context-aware systems are “systems that are able to adapt their operations to context changes without explicit user intervention” [6, 15]. Their aim is to increase the usability and the effectiveness of the software systems by taking into account the environmental context. Matthias et al. presented a layered conceptual design architecture [6] (Figure 10-A). This architecture contains the different elements that are common to most existing context-aware systems [16-20]. Layer one at bottom is the hardware/software sensors to detect the changes in the environment, and then these changes are retrieved from the sensors at layer two. Reasoning on and interpreting the gathered context data to get higher level context information is performed at layer three. Layer four organizes the context information and offering them to the functional system. Finally, layer five represents the functional system that has the system core functionality and the actual reaction to context changes, where the required adaptation actions are decided and acted on the functional system.

Self-adaptive software systems are “systems that modifies its own structure and/or behaviour in response to changes in its requirements and/or operating environment” [21-23]. An abstract layered architecture for self-adaptive systems is presented by Kramer and Magee (Figure 10-B) [3]. Layer one (i.e. the bottom layer) consists of the set of interconnected components that represents the system core functionality and the resources that is needed for its operation. It includes facilities to (a) report the current components status to the higher layers, and (b) support component(s) creation, deletion and interconnection. Layer two responses to the changes coming from the component control layer, which executes one of the pre-computed plans to cope with these changes. Layer three takes the state of running system and the specification of the high-level goals and produces a plan(s) to maintain these goals when one of them is violated or new goals are introduced.

![Figure 10: Context-aware and self-adaptive software systems abstract layered architectures](image)

5.2 Research on Context-aware and Self-adaptive Systems

The context-aware and self-adaptive systems have a common goal of how to adapt the system in response to context changes. However, in research they have different emphases. Firstly, context aware systems [16-17, 19, 24] concentrate on modelling the context information, processing the low level retrieved data to infer high level context information, and then this information is stored to be used by the functional system (i.e. sensing and processing operations). However, they hardwired the...
deciding and acting operations into the functional system. Consequently, the system ability is limited to only cope with a fixed set of context changes that has been built in during the system design time. Furthermore, this research is not usually concerned with requirements changes. Figure 11-A shows the state of existing context-aware systems frameworks, where they (a) separate the context model form the functional system and its management; (b) have an explicit process for sensing and processing the context information and with less attention to the deciding and acting aspects to cope with anticipated and unanticipated context changes.

Secondly, a number of frameworks and middleware have proposed to facilitate the development of self-adaptive software systems [7, 21-23, 25-32]. These frameworks consider the four operations of the control loop to make the system able to change its structure and/or behaviour in response to context/requirements changes. However, they hardwire the context processing/management with the functional system and its management. As such, the system complexity is increased, and the reasoning on the context data to infer the context information cannot be performed. Figure 11-B shows the state of existing self-adaptive systems frameworks, where they (a) separate the system management from its functionality; (b) consider the system environment context implicitly and this increases the system complexity and limits the system ability to process and manage the context information.

The abstract layered architecture proposed by Kramer and Magee can be used as a base for developing context-aware adaptive systems [3]. However, their architecture does not consider modelling and processing the context information explicitly (i.e. context aware systems view). The context-aware systems layered architecture supports the later issue [6], but it is limited in how the system can adapt itself in response to context changes. In addition, both architectures do not have the system and its runtime representation layer to make the system able to cope with unanticipated context changes. To the best of our knowledge there is no previous work that addresses how to have an explicit context representation with close corresponding self-adaptivity.

Our proposed approach integrates the context-aware and self-adaptive systems perspectives to form what we call context-aware adaptive software system. To maintain the context-awareness view we modelled the context separately at layer two and then it is processed and managed at layer three. Our approach has the four operations of the control loop (i.e. monitoring and acting at layer two, and deciding and analysing at layer three) to incorporate the self-adaptive systems perspective. Finally, the system and its context runtime representation at layer two enable the system’s runtime change to cope with anticipated and unanticipated changes in the system’s requirements and environment while the system is in operation.
6 Conclusions and Future Work

Context-awareness and self-adaptivity are treated separately in existing research. But real world demands their consideration in an integrated manner as shown in our motivating scenario. In this paper, we have presented our layered architecture that can be used for developing context-aware adaptive software systems. To realize the approach, we used our component model that explicitly supports in the component interfaces the definition of the required and provided functionalities, context information and management actions. In addition, our developed tool supports the generation of the context-aware adaptive software system implementations from their models. The tool also supports the visual validation and the formal verification of the systems’ context-aware adaptive behaviour through a graphical user interface.

Our layered architecture, the system and its context components, the system and its context representation, and the change management, partition the system in a way that allows the context-aware adaptive systems requirements to be handled using different layers. First, the context information is (a) modelled at layer two and (b) processed and managed at layer three. Second, the change management layer (i.e. layer three) is responsible for adapting the system to cope with context and requirement changes. Third, the monitoring and acting operations at layer two captures the system-context operational relationships. In addition the whole control loop at layers two and three is used to capture the system-context management relationships.

There are several future directions for this research. First, in this report we used our component model and its tool support to realize our architecture. Our realization approach has a runtime representation of the system aspects to be able to cope with the unanticipated context changes. As such a runtime management interface will be developed to fully realize this capability. Second, we will identify a set of design patterns that can be used to cope with the unanticipated context changes without the system designer involvement to define the required changes manually. Third, our tool will be extended to support the runtime activities of the context-aware adaptive software systems.

7 References