Second Review Report

A Model-based Approach to Developing Context-aware Adaptive Software Systems

Date of Submission: 21-2-2011

Submitted by: Mahmoud Hussein

Supervisors: Prof. Jun Han and Dr. Alan Colman

Committee members: Prof. Chengfei Liu, and Dr. Jinjun Chen
# Table of Contents

1. Introduction ............................................................................................................................ 1

2. Motivating Scenarios and Requirements Analysis ................................................................. 1

3. Literature Review and Research Questions ........................................................................... 2
   3.1 Context Modelling and Management ............................................................................... 2
   3.2 Functional System Runtime Adaptivity ........................................................................... 3
   3.3 System-Context Relationships ....................................................................................... 4

4. General Approach .................................................................................................................. 4

5. Current Progress ..................................................................................................................... 5
   5.1 Survey on Context-aware Adaptive Systems ................................................................. 5
   5.2 An Architecture Approach to Developing Context-aware Adaptive Systems ............... 6
   5.3 An Approach to Modeling and Realizing Context-aware Adaptive Systems ............... 6
   5.4 An Approach to Validating a Context-aware Adaptive System Behavior ...................... 7

6. Approach Validation .............................................................................................................. 8

7. Publication Plan ..................................................................................................................... 9

8. Schedule ................................................................................................................................. 9

9. Thesis Structure ................................................................................................................... 10

Appendix A ................................................................................................................................ 11

References .................................................................................................................................. 12
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 model, design, verify, and realize such systems that evolve at runtime [1-5].

We focus in this research on software systems that need to cope with runtime context changes (i.e. changes in or to their operating environments), which we call context-aware adaptive software systems. The context changes that need to be taken into account by a software system can be (a) anticipated changes, where these changes are known beforehand at design time, and then the system should have a set of predefined reactions (i.e. system adaptations) to them [3]; (b) unanticipated changes, where the system is usually deployed in an environment that is not totally anticipated at the design time, i.e. experiencing changes unknown a priori [6]. To cope with the unknown context changes, the system should have the ability to incorporate new context information and corresponding system adaptations to it at runtime.

For a system to have the ability to cope with anticipated and unanticipated context changes, it needs to have (a) an explicit representation of its context that is able to be changed at runtime to incorporate new context information; (b) an ability to change its structure and/or behaviour (i.e. runtime system adaptability) to cope with the context information change; (c) a management mechanism that makes decisions about the system changes as reaction(s) to context changes. In addition, this management mechanism should be changeable at runtime to incorporate new adaptive behaviour(s) into the overall system.

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 [7-9]. 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 (unanticipated) 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 context (and other) changes by separating the system functionality from its management and has paid less attention to how the context is represented, processed, managed, and made available to the system. As a consequence, the system complexity is increased by hardwareing the context processing and management operations together with the system functionality and its management. Furthermore, there is very little consideration concerning unanticipated context changes.

This research’s objective is to develop an approach to developing software systems that are able to cope with anticipated and unanticipated context changes to achieve and/or preserve their goals while the systems are in operation.

2. Motivating Scenarios and Requirements Analysis

A context-aware vehicle is equipped with a set of software applications to navigate and organize the driver’s affairs while on the road. The applications in this vehicle include the route planner application, which assists the driver with route planning and congestion avoidance. We present below a number of scenarios where the route planner application is used. They will serve to (1) help the discussion of the report; (2) identify the requirements that need to be considered in developing a software system that has the ability to cope with anticipated and unanticipated context changes.

**Scenario 1:** The driver needs to plan his journey home, and he turns on the route planner that senses/receives all the available context information. The driver route preference is available through his mobile, and the traffic information can be obtained from roadside units. With this information, a suitable route planning algorithm is selected that takes into account the available context information. Then, the route planner shows the available routes to the driver based on the current context information. The driver chooses a suitable route and starts his journey home. During the journey, the vehicle routes are recalculated every five minutes (driver requirement) to take into account the changes in the traffic information (e.g. congestion has formed en-route). Later, due to loss of communication with roadside units, the system uses another route planning algorithm which does not take into account the traffic information.

**Scenario 2:** During the life time of the vehicle route planer application, a new type of context information source may become available, such as up-to-minute information about the surrounding vehicles in the driver’s lane and neighbouring lanes. For the system to take account of the new context information and suggest lane changes, it needs to (a) change its context model (i.e. adding the new context information); (b) add an algorithm that considers the surrounding vehicles en-route when performing the route planning; (c) add new adaptive behaviours to guide the switching between the currently used route planning algorithms and the new one depending on the availability of the context information. These system changes need to be performed while the...
system is in operation without sending the vehicle to a garage (i.e. a remote software upgrade) to save the driver’s time and reduce the cost of doing the software upgrade at the garage.

The above scenarios show a number of general requirements for the system’s context-awareness and adaptivity:

Requirements for context modelling and management: First, the system has a large amount of environment information about the driver, the vehicle, and the vehicle environment (e.g. the nearby vehicles, the traffic information, etc.) which affects the system operation. As such, the context information needs to be modelled explicitly to reduce the system complexity that can be caused by hardwiring the context information processing and management with the functional system and its management.

Second, during the system operation the context model changes. Some of these changes can be anticipated. For example, one route planning algorithm needs only the driver route preference as a context model. As such, only the context acquisition entity for driver route preference is enabled and the other context acquisition entities (e.g. those for traffic information) are disabled. On the other hand, some context model changes can be unanticipated where a new type of context information is added to the context model at runtime (e.g. the surrounding vehicles on-route). Consequently, the runtime representation of the context model should support adaptation to cope with anticipated context changes and evolution to incorporate new context entities.

Requirements for functional system runtime adaptability: First, in response to anticipated context changes, the functional system needs to adapt itself. For example, the selection of route planning algorithms is based on the available context information, where the route planning algorithm used when the system has only the driver route preference is different from the one used when the traffic information is also available. As such, the functional system needs to have the ability to adapt itself to cope with the anticipated context changes.

Second, the functional system not only should have a response to the anticipated context changes, but also the unanticipated context changes by evolving itself. For example, the addition of the information about surrounding vehicles on-route (i.e. new context information) to the context model raises the need in the functional system for a new algorithm that considers this new context information in computing the possible routes and in suggesting lane changes. As a consequence, the functional system should be able to cope with the unanticipated context changes.

Requirements concerning the system-context relationships: First, the context information can be operational or management context. The operational context is needed by the functional system to continue its operation. For example, the vehicle location is needed by the route planning algorithm to operate correctly. On the other hand, the management context causes the system to adapt form a system’s configuration and/or behaviour to another. For example, the unavailability of the driver route preference causes the system to adapt from a route planning algorithm to another. Consequently, the two types of relationships between context changes and system reactions need to be considered: operational and management system-context relationships.

Second, the system needs to have changeable system-context relationships. For example, when a context entity is disabled, the system-context relationships depending on it are not needed. For example, when the traffic information is not available, the relationships between the traffic information context entity and the route planning algorithms and that between the traffic information and the system management strategy, are not required to be active. As such, these relationships should be disabled to reduce their overhead on the system. In addition, to cope with the unanticipated context changes, the relationships between the system and its context need to be evolved at runtime to consider the new context information by introducing new system-context relationships.

Third, to achieve runtime changes to the system-context relationships and reduce the system modelling complexity (in hardwiring these relationships with the functional system), these relationships need to be explicitly modelled. Finally, when such system-context relationship changes happen, there is a need to validate/verify the integrity of the relationships, i.e. not introducing errors.

3. Literature Review and Research Questions

In the previous section, we have identified a set of requirements that need to be considered in developing a software system that is able to adapt/evolve itself in response to context changes. In this section, we analyze in brief existing approaches with regard to these requirements and identify the research gaps (Figure 1 in appendix (A) shows a summary of how existing approaches address the requirements, and more details can be found in [10]).

3.1 Context Modelling and Management

Explicit context model: Existing research considers the context in two ways. First, the context is considered implicitly in some approaches, i.e. the context and its management being modeled as a set of variables in the system [11-16]. When there are a large number of these context variables, the system complexity is increased
and the context management operations cannot be performed easily (including reasoning, adaptation, etc.). Second, an explicit context model is maintained at runtime in other approaches to reduce the system complexity and enable the context model management operations [17-20]. In this research, we maintain an explicit context model at the runtime to retain the relevant benefits.

**Context model adaptation:** The context model consists of a set of entities. During the runtime, a subset of these entities need to be selected (enabled) and used by the system and the other context entities are disabled to reduce the context monitoring overhead [1, 5]. The context model entities selection (i.e. the context entities enabling and disabling) depends on the availability of their context providers and the functional system state. For example, when the driver selects a route planner algorithm, its required context entities are enabled and the other entities are disabled. In the existing literature, only limited work supports structural changes to the context model [21-22]. These approaches only perform the context entities selection during the system deployment time based on the context providers’ availability. However, during the runtime the context providers can come and go and the functional system state changes affect the context model structure. As such, an approach is needed to enable the changes to the context model structure at runtime to reduce the context monitoring overhead.

**Context model evolution:** The context model structure changes can be anticipated (i.e. known at the design time) or unanticipated (e.g. new context information is introduced at runtime). Existing context modeling approaches do not consider how to cope with the unanticipated structure changes to the context model while the system is in operation [7-8]. An approach has been introduced to enable the system management changes [23]. In this approach, the addition of a new adaptation rule may make the system be able to take into account new context information. However, they model the context information as a set of variables inside the functional system management (i.e. using an implicit context model). As such, the context model entities relationships (e.g. the nearby relationship between two vehicles) and their dependencies (e.g. the distance between two vehicles depends on their speeds) cannot be captured. In addition, considering the context information implicitly with the system management increases its complexity, and the context information processing and management become difficult (e.g. inferring the high level context). As a consequence, an approach is needed to make the context model able to incorporate new context information at runtime while considering the above aspects.

**Research Questions:**

**RQ1.1:** How to enable context model entities selection at runtime to reduce the context monitoring overhead?

**RQ1.2:** How to make the context model able to incorporate new context information at runtime?

### 3.2 Functional System Runtime Adaptivity

**Runtime functional system changes:** In response to context changes, the functional system needs to adapt itself [24]. The functional system adaptations (changes) can be known at the design time (i.e. anticipated), or unknown till the runtime (i.e. unanticipated). Existing approaches consider the anticipated functional changes, where the system’s possible adaptations are specified at design time. Then, these specified adaptations are used at runtime to change the system to preserve and/or achieve its goals [25-33]. However, they do not support the unanticipated functional system changes (i.e. the functional system evolution) to cope with the unanticipated context changes when the already specified adaptations are not suitable (e.g. introducing new functional system elements while the system is in operation). Consequently, there is a need for an approach that enables the unanticipated functional system changes in response to the unanticipated context changes.

**Runtime functional system models:** To enable the unanticipated functional system changes, a model of the functional system’s definitional aspects (i.e. the definitions of the system’s components and connectors) needs to be maintained at runtime. This model should be changeable to enable the definition of new functional system elements at runtime. Some of the existing approaches maintain a runtime model of the system [11, 15, 20]. But, their models maintain only the system state (to initiate the adaptation process in case of system goals violation) and not the system’s definitional aspects. In addition, these models are not changeable at the runtime.

**Realization of unanticipated functional system changes:** To realize the unanticipated functional system changes, the running functional system need to be updated by incorporating new functional elements at runtime. To do so, a model driven engineering approach is promising. However, the existing model driven engineering approaches are used to develop static systems with less attention to dynamic systems such as the context-aware adaptive systems [34-36]. These approaches are used to generate the system implementations from their models at design time without considering the system update while it is in operation. As such, there is a need for a model driven engineering approach that supports the realizations of the unanticipated functional system changes.

**Research Questions:**

**RQ2.1:** How to maintain the functional system model at runtime and enable its update by incorporating the definitions of new functional system elements?
RQ2.2: How to develop a model driven engineering approach that supports the realizations of unanticipated functional system changes (i.e. the functional system runtime update)?

3.3 System-Context Relationships

Explicit representation of the relationships: The system and its context are related entities where context changes affect the system operations. The system-context relationships can be classified into two types depending on the context effect on the system: operational and management relationships. In large scale systems, there may be a large number of system reactions in response to the context changes, and then the process of capturing the system-context relationships is complex. Most of existing research models the system-context relationships implicitly [10]. As such the system complexity is increased, and the modeling of these relationships becomes complex and error prone. Some approaches model only the management relationships explicitly. But, they hardware the context model with the relationships model. As such, the relationships’ modeling complexity is increased (e.g. [11, 13, 23]). Consequently, there is a need for an approach to capture the system-context relationships explicitly while considering the relationships modeling complexity.

Runtime relationship changes: Existing techniques do not support the runtime changes to the system-context relationships except [23]. In their approach, the management relationships are represented as a component based system that can be changed to incorporate new relationships. However, they do not represent the operational relationships explicitly and do not enable their changes (i.e. they are more concerned with the system management). In addition, they do not support the runtime management of the relationships to reduce their overhead into the system (e.g. the removal of unwanted relationships), where their management relationships changes are performed manually. As a consequence, an approach is needed that enable the system-context operational and management relationships changes while the system is in operation.

Validation of management relationships: Some of the existing approaches use state-based models (e.g. Petri Net or Labeled Transition System) [12-13] to represent the management relationships. These techniques enable the validation of the relationships to assure their correctness. However, the validation process cannot be effectively performed when a large number of adaptation behaviors are involved because of the state explosion problem. In addition, the existing techniques do not identify the possible errors that need to be detected (i.e. the errors that can appear in the management relationships). Furthermore, existing approaches do not consider the runtime validation of the management relationships when new relationships are introduced at runtime. As such, an approach is needed to validate the management relationships at design time and at runtime, in particular when there are a large number of these relationships.

Research Questions:

RQ3.1: How to capture the system-context operational and management relationships?
RQ3.2: How to enable the system-context relationships runtime adaptation to reduce their overhead (RQ3.2.1) and evolution to incorporate new relationships (RQ3.2.2)?
RQ3.3: How to validate the management relationships at design time (RQ3.3.1) and at runtime (RQ3.3.2)?

4. General Approach

In order to develop a software system that is able to cope with anticipated and unanticipated context changes by adapting its context model, functional system and system-context relationships, a system development process is shown in Figure 1. We will follow this process in our research and it is divided into two major stages. First, the design stage is used for modelling a system that has the ability to cope with the context changes that are known beforehand, and then the system implementations are generated from their models (i.e. steps one and two). Second, the runtime stage starts when there is a running instance of the system. In this stage, a capability is added to the system for making it able to incorporate new context information and the system reactions to it (i.e. steps three and four).

Step 1: To enable the integrated modelling of the system and its contexts (i.e. enabling the software engineer to model a context-aware adaptive software system from its requirements), a modelling approach will be developed to enable the capturing of the context model, the functional system model, and the system-context operational and management (i.e. the adaptations model) relationships. The adaptations model provides a way to enable the adaptations of the context model, the functional system, and the system-context relationships (RQ1.1, RQ3.1 and RQ3.2.1).

Step 2: To validate the system adaptive behaviour at the design time, an approach will be developed which transforms the system adaptations model to Petri Nets for further formal validation to detect errors such as inconsistency, redundancy, etc. In addition, to support the detection of missing and incorrect adaptation behaviours, a visual validation approach will be provided (RQ3.3.1). Then, a model driven engineering approach will be developed to enable the system implementations generation from their models.

Step 3: To incorporate new context information and the system reaction(s) to it, two approaches will be developed. First, a mechanism will be developed to enable the changes to systems models manually at runtime,
where the context model is modified by adding the new context information, and then the required changes into the functional system and the adaptations model are performed accordingly. Second, a set of design patterns will be identified to perform the required changes to system models automatically (RQ1.2, RQ2.1 and RQ3.2.2).

Step 4: An approach will be developed to enable the system adaptive behaviour validation at runtime to assure that the new added adaptive behaviours do not cause problems (e.g. conflict) with the already specified adaptive behaviours (RQ3.3.2). Then, a model driven engineering approach will be developed to enable the system implementations update and their synchronization with the running system for making it able to cope with the new context situations that were not considered beforehand (RQ2.2).

5. Current Progress

During the last one and half year, four tasks have been completed. First, we have performed an analytical survey into context-aware systems and self-adaptive systems to identify the research gaps [10]. Second, in order to structure our work, we have proposed an architectural approach for context-aware adaptive systems in general. This architecture serves as a roadmap for this research and the extension of it (e.g. making the system able to cope with the requirements changes) [37]. Third, we have developed an approach for modeling and realizing the context-aware adaptive systems as the step one discussed above [38]. Finally, we have investigated the system adaptive behavior validation (i.e. step two) by transforming the designed adaptive behavior model to a Petri Net, specifying the errors that need to be detected using temporal logic, and then the Romeo tool is used to perform the validation [39].

5.1 Survey on Context-aware Adaptive Systems

In relation to our research area, there are several surveys that have been carried out from different viewpoints such as surveys on context modelling techniques, system runtime adaptivity, context-aware systems, and self-adaptive systems. However, these surveys did not systematically consider the requirements for integrating context-awareness and self-adaptation (i.e. our focus). Consequently, we have carried out an analytical survey of research into context-aware adaptive software systems from both of these perspectives, with a particular focus on issues concerning their integration [10] (Figure 1 in appendix (A) shows the survey results).

**Context modelling:** Bettini et al. [8], Bolchini et al. [40], and Strang et al. [41] surveyed and evaluated the proposed context modelling techniques. Those authors have specified a set of requirements for modeling the context as a separate entity from the system, without pay attention of how the context information affects the system operations. In this research, we concentrate on the context modeling requirements from the perspective of how the context affects the system operations and/or adaptations. As such, in addition to the traditional context modeling requirements, we present a set of requirements that were not considered in existing context modeling surveys such as (1) the social context model; (2) the context model subjectivity; (3) the predication of the context trends; (4) the context model runtime representation to enable its adaptation and evolution.

**System runtime adaptivity:** Several authors discussed the requirements for the traditional [42-44] and adaptive [45-47] systems modelling. But our concern is the adaptive systems modelling aspects in relation to its context. In particular, how the system’s multiple models are managed during the software execution to enable its adaptation and evolution. Consequently, we have identified the system’s solution context, runtime models, and evolution as new requirements that were not considered in the existing surveys.

**System-context relationships:** In the area of context-aware systems, Matthias et al [7] identified the requirements that context-aware systems should have, and then an evaluation for research in this area is
provided. But they did not consider the requirements concerning the context effect into the system, where this type of research is more concerned with how to model, process, and manage the context information. Research in self-adaptive systems is more concerned with how the system responds to its context and requirements changes with less attention to how the context is modelled and processed inside the system. In this respect, Salehie et al. [1], Huesorscher et al. [5], Nami et al. [48], and Bradbury et al. [47] have surveyed the existing self-adaptive software systems without considering the requirements for integrating the context-awareness and self-adaptation. In our survey, we have identified new requirements that are related to the system-context relationships such as (1) the relationships explicit representation; (2) the management relationships validation to assure the correctness of the system adaptive behaviour; (3) the runtime adaptation and evolution of these relationships while the system is in operation.

5.2 An Architecture Approach to Developing Context-aware Adaptive Systems

To integrate the context-awareness and self-adaptivity perspectives and fill the research gaps that are identified in our survey, we have proposed an architectural approach that can be used to develop context-aware adaptive software systems [37]. Our layered architecture is shown in Figure 2, and has three layers as follows:

The functional system and its context (layer one at the bottom of Figure 2): This layer has three elements. First, the functional system consists of a set of components that are used to achieve the system’s core functionality (e.g. C1, C2, and C3). Second, the context contains a set of environment entities that affect the system’s operations and/or adaptations (i.e. EC1, EC2, and EC3). Finally, there are the interfaces with the management layers (i.e. layers two and three). First, these interfaces contain a set of sensors/monitors to detect the changes in the system or its context, and then they inform the management layers by these changes to take the required adaptation actions. Second, these interfaces also have a set of effectors to apply the adaptation actions, which have been decided by the management layers, into the running system.

The runtime representation of the system and its context (Layer two): This layer has a representation of the system and its context and has two types of operations. The context model captures and organizes the environment context information. The system models capture and maintain the system state, including such aspects as the system’s structure and behaviour models. The monitoring/sensing operation is responsible for keeping the context model and system models up-to-date. The acting operation is used for applying the adaptation actions that are coming from layer three to the running system and to the context model. For adapting the running system, the adaptation actions are applied to the running system models by the acting operation 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 by its effectors. The context model is adapted by the acting operation, where the inferred high level context information by the processing operation is stored into the context model. In addition, based on the functional system state, a set of context model entities are selected by the deciding operation, which they are needed by the functional system to continue its operation.

Change management (Layer three): This layer maintains a precise specification of the system’s functional requirements and the non-functional requirements such as performance, security, and etc (i.e. the requirements models). In addition, it has three types of 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 the case of consistency violation. Second, the processing operation is used for inferring the high level context information. Finally, the deciding operation determines the required adaptation actions in response to context and/or requirements changes. There are a set of pre-defined adaptation actions in response to an expected class of state changes (i.e. the adaptation scripts). When none of the available adaptation actions are suitable to cope with these changes, the deciding operation tries to generate new adaptation actions on-the-fly. If the generation of new adaptation actions is not possible, the system administrator is informed to perform the required changes.

5.3 An Approach to Modeling and Realizing Context-aware Adaptive Systems

Our meta-model for context-aware adaptive software systems is shown in Figure 3 and more details about it can be found in [38]. 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 functional system model (i.e. functional system) and their management relationships (i.e. management). The meta-model is particularly designed to address the research gaps we need to tackle in step one.

1. Separate the context model from the functional system model. Our meta-model (see Figure 3) represents the context model and the system model as two separate but related composites. This separation reduces the system modelling complexity, where the two aspects (i.e. the functional system and its context) and their relationships can be clearly captured. This separation of concerns also increases the models’ maintainability and runtime evolvability as it makes them easier to change and with less chance of making errors compared to the case of hardwiring the context model in the functional system model.

2. Enable the runtime adaptation of the context model (RQ1.1), the functional system, and their relationships (RQ3.2.1) (i.e. the system elements). To enable the runtime change of the system elements in response to 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 3). 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 during execution, including the component state field of the functional system and the entity attribute of the context model in Figure 3. Examples are the response time of a functional system component and the current location of the vehicle. The execution path(s) are represented as a set of rules (i.e. the management composite) that guide the transition between the different states of the system elements in Figure 3. Based on the rule condition(s) (e.g. vehicle speed is greater than 70 km/h), a set of adaptation actions are triggered, which are for adding, removing, replacing, or modifying the system elements (e.g. use route planning one).

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). This includes the separation of the functional system component definition from its realization in the functional system composite, the entity attribute definition from its realization (i.e. context provider) in the context model composite, and the adaptation action definition from its realization in the change management composite (see in Figure 3). Doing so enables the change or swap of the element realization at runtime without affecting the system’s overall structure and its element 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 unanticipated changes.

3. Capture the operational and management system-context relationships (RQ3.1). As shown in Figure 3, 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 change management 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 change management, which in turn selects a specific realization for the route planner component.

5.4 An Approach to Validating a Context-aware Adaptive System Behavior

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 at the same time. 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 being fired repeatedly means that there is a cycle within the adaptation rules. Finally, if there is a context situation that has no reaction from the system but supposed to have, then the system adaptive behaviour is incomplete. As such, the system adaptive behaviour needs to be validated to detect these errors (Step 2 discussed above).

To validate the system adaptive behaviour using the Romeo model checker tool [49] (RQ3.3.1), first we transform the specified adaptive behaviour model to a Petri Net [50]. Second, we represent the possible errors using Timed Computation Tree Logic (TCTL) [51]. For example, the formula “EF[0,0]M(3)+M(6)>1” means that there exist an execution path where the marking of the Petri Net places 3 and 6 are greater than 1 (i.e. both places are active in the same time). This formula can be used for checking the inconsistency between the adaptation actions, where two inconsistent adaptation actions are triggered simultaneously.

The first step to build a Petri Net model is to enumerate its states, which correspond to the possible variations (i.e. configurations and behaviours) of the system during the runtime. Form the system re-configurations point of view, the number of the system states can be calculated as the product of the possible variations of the system components themselves. For example, assume that we have a system that consists of 10 components. First, two components of them can be added and removed. Second, another two components have three variants (one of these variants is selected based on the system requirements at runtime). Finally, the other six components are fixed which represent the system basic functionality. As such, the number of possible variations of the system is 36 (i.e. $2^2*3^3$). Furthermore, if we consider the case where another component (from the six fixed components) has three variants, then the total number of system states (variations) is increased to 108 (i.e. $3^3*36$). Consequently, the number of the system states grows exponentially with the possible system adaptations.

In our approach, we reduced the number of the system model states by considering a system state as a combination of multiple sub-states of the system components variations (i.e. not the whole system configuration as one state). In addition, we also take into account the components dependencies. We group the possible system variations into two types: independent and dependent variations as to whether or not the system component change is dependent on the other components. Then, for each dependent variation group, a state model is constructed. The number of states in the constructed model equals to the product of this group members variations. Finally, state models are constructed for each independent system component variation (i.e. a model of two states in case of component addition and removal or a model of n states where n is the number of possible component variants/replacements).

In our Petri Net model of the system adaptive behaviour, the adaptation conditions are represented as input places and the adaptation actions as output places. The Petri Net transition links the input and output places, where the evaluation of a condition(s) to true actives an adaptation action(s). The input places (i.e. the adaptation conditions) can be shared between multiple adaptation behaviours, and then we consider them as output places too. As such, each transition keeps its input places active to be used in other adaptation behaviours. But the adaptation rules can be activated several times, and then we add the transition enabling condition input place for making the rule evaluation to be true once. This way of transforming the system adaptive behaviour to Petri Net reduces the number of the system states. We transform each single adaptation action (i.e. single component change) into an output place, and then the whole system state can be inferred using multiple output places (i.e. the sub-states we mentioned above for reducing the state space) and not a single place. Details of our approach can be found in [39].

6. Approach Validation

Our approach has two stages: design time and runtime, and then we need to have a set of metrics for evaluating our approach at both stages. During the system design time, the following set of metrics will be used to compare (evaluate) our modelling approach with respect to the existing techniques (obtained from [52-54]):

1. Engineering effort: This metric measure the time and cost needed to develop an adaptive system using our approach with regard to using a traditional approach (i.e. the development time and cost reduction when the new approach is used).

2. Number of automated administrative tasks: The adaptive systems are proposed to automate some tasks that were to be performed manually. Therefore, the number of tasks that are automated by the system management (i.e. do not need the system’s user/administrator involvement) need to be measured with regard to the total number of tasks require the system’s user/administrator involvement.

3. Separation of concerns index: This metric is an indication of (a) the dependency between the context model, the functional system, and the system management; (b) the separation of the system management operations (i.e. monitoring, analysis, planning, and execution).

4. Factors influencing the system adaptivity: The system and its environment states need to be taken into account by the system management, and then this metric show to what degree they are considered.
After developing a context-aware adaptive system, a number of metrics need to be measured during its operation to assure that the developed system benefits of adding the adaptivity feature is more than the system adaptivity overhead such as:

5. Responsiveness:
   - Working time vs. adaptivity time: The CPU time spent by the system to decide and act the required system adaptation with regard to the time spent to perform its core functionality.
   - Response time: The system response time variation in the presence of the system adaptivity with respect to the case where no adaptivity is introduced.

6. Memory usage: The amount of memory usage that is increased by adding the adaptivity to the system.

Finally, in our work we perform a formal validation of the system adaptive behaviour. We need to measure the system state-model reduction (our work has this feature) relative to the existing approaches. In addition, we need to perform the system adaptive behaviour validation at runtime. As the system is in operation, the verification should be performed quickly, and then the performance (i.e. the execution time) of our technique needs to be measured to assure its applicability.

7. Publication Plan

The schedule for publishing the completed four tasks is as follows.

First, a paper reporting the approach developed for modeling and realizing the context-aware adaptive software systems was submitted to the 35th IEEE Conference on Computer Software and Applications (Munich-Germany, July 18-22, 2011)[38].

Second, the analytical survey into context-aware adaptive systems will be submitted to the ACM Transactions on Autonomous and Adaptive Systems [10].

Third, the approach for validating the system adaptive behavior will be submitted to a suitable venue (we were planning to submit it to the ACM SIGSOFT Symposium on the Foundations of Software Engineering) [39].

Finally, we will submit a paper presenting the architectural approach for context-aware adaptive systems to the IEEE Software Magazine (we already sent the abstract, and the result was positive about the article suitability for the magazine) [37].

8. Schedule

The work during the PhD period is scheduled as in Table 1. During the first nine months, a literature review has been completed and then an analysis of existing approaches was presented as a technical report [10]. While we was doing the analysis, we have developed an architectural approach for context-aware adaptive systems [37]. Following the literature review, we follow the two phases (discussed in section four) for developing a software system that is able to cope with anticipated and unanticipated context changes.

In the design time phase (around twelve months), we need to develop an approach to modelling and realizing software systems that are able to cope with context changes that are known beforehand at deign time. In this regard, we have developed (a) an approach to model the system, the context, and their relationships explicitly [38]; (b) a tool to generate the system implementations from their models [55]; (c) an approach to validate the system adaptive behaviour [39].

In the runtime phase (around twelve months), we use the models and the system realizations that have been constructed during the first phase to enable the system runtime evolution. In this phase, approaches will be developed for (a) making the system able to cope with the unanticipated context changes; (b) assuring the system adaptive behaviour correctness when new adaptive behaviours are introduced at runtime. Finally, thesis writing will take around six months.

Before we start phase two, we need to refine some of the already completed tasks in phase one (around six months). The refinements are about the proposed approach and its empirical validations.

<table>
<thead>
<tr>
<th>Table 1: PhD activities schedule and their allocated periods</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Allocated Time Periods</strong></td>
</tr>
<tr>
<td>Literature review</td>
</tr>
<tr>
<td>Architecture for Context-aware Adaptive Systems</td>
</tr>
<tr>
<td>Integrated system and context modeling</td>
</tr>
<tr>
<td>System adaptive behavior verification</td>
</tr>
<tr>
<td>Define the work that has been done</td>
</tr>
<tr>
<td>Enabling the System runtime evolution</td>
</tr>
<tr>
<td>System adaptive behavior runtime verification</td>
</tr>
<tr>
<td>Thesis writing</td>
</tr>
</tbody>
</table>
9. Thesis Structure

The thesis is structured into three parts. The first part consists of an introduction to the context-aware adaptive systems, an analysis of existing approaches, and an architectural approach for developing context-aware adaptive systems. Our proposed approach is introduced in the second part. This approach is used to develop software systems that are able to cope with anticipated and unanticipated context changes. Finally, the third part contains our developed tool, the approach validation, and the conclusions and future work.


Chapter 1: Introduction
1.1 Context-aware Adaptive Software Systems
1.2 Research Motivation
1.3 Research Problem
1.4 Research Contributions
1.5 Thesis Organization

2.1 Requirements for Context-Aware Adaptive Systems
2.1.1 Requirements for Context Modelling
2.1.2 Requirements for System Runtime Adaptivity
2.2.3 Requirements concerning the System-Context Relationships
2.2 Analysis of Context-aware Adaptive Software Frameworks
2.2.1 Frameworks for Context-aware Adaptive Systems
2.2.2 Frameworks for Adaptive Systems
2.2.3 Frameworks for Context-aware Systems
2.2.4 Frameworks Summary
2.3 Summary

3.1 Context-awareness and Self-adaptivity
3.1.1 Context-aware and Self-adaptive Systems
3.1.2 Context-awareness and Self-adaptivity Research
3.2 Context-aware Adaptive Software System Architecture
3.2.1 Context-aware Adaptive Software Systems Control Loops
3.2.2 The Proposed Layered Architecture
3.3. An Illustrative Example
3.3.1 Functional System and its Context
3.3.2 Representation of Functional System and its Context
3.3.3 Change Management
3.4 Summary

Part II: Model-based Development of Context-aware Adaptive Software Systems

Chapter 4: An Approach to Modelling and Realizing Context-aware Adaptive Systems
4.1 The Approach
4.1.1 Context-aware Adaptive Software System Meta-model
4.1.2 Context-aware Adaptive Systems Component Model
4.1.3 Validating the System Adaptive Behaviour Visually
4.2 Illustrative Example
4.3 Summary

Chapter 5: Verifying the System Adaptive Behaviour at Design Time
5.1 Building the System Adaptive Behaviour State-based Model
5.2 The System Adaptive Behaviour Errors
5.2.1 Inconsistency
5.2.2 Redundancy
5.2.3 Circularity
5.2.4 Incompleteness
5.3 Illustrative Example
5.4 Summary

Chapter 6: Enabling the System Runtime Evolution
6.1 The Approach
6.1.1 Context Model Evolution
6.1.2 Functional System Evolution
6.1.3 System Adaptive Behaviour Evolution
6.2 Illustrative Example
6.3 Summary
Chapter 7: System Adaptive Behaviour Runtime Verification
  7.1 An Incremental State-based Model for the System Adaptive Behaviour
  7.2. An Approach for the Runtime Verification
  7.3 Illustrative Example
  7.4. Summary

Part III: Implemented Tool, Validation, and Conclusions
Chapter 8: Tool Support
  8.1 CAST: Context-aware Adaptive Systems Development Tool
  8.2 The Architecture of the Tool and its Design
  8.2 Tool Inputs and Outputs
  8.3 Tool Examples
  8.4 Summary

Chapter 9: Validation
  9.1 Case Studies
    9.1.1. Modelling and Realization Approach
    9.1.2. Verifying the System Adaptive Behaviour at Design Time
    9.1.3. Enabling the System Runtime Evolution
    9.1.4. System Adaptive Behaviour Runtime Verification
  9.2 Evaluating our Approach at Design Time
  9.3 Evaluating the System Developed by using our Approach at Runtime
  9.4 Summary

Chapter 10: Conclusion and Future work
  10.1 Conclusions
  10.2 Future Work

Appendices:
  Appendix A: Java Code of our Tool
  Appendix B: Java Code for the Tool Examples

Appendix A
An analytical summary of existing approaches for developing context-aware adaptive software systems is shown in Figure 1, and more details can be found in [10].

Figure 1: Summary of existing approaches for context-aware adaptive systems
References


