Faculty of Information and Communication Technologies

Third Review Report

A Model-driven Approach to Developing and Evolving Context-aware Adaptive Software Systems

Date of Submission: 7-3-2012

Submitted by: Mahmoud Hussein

Supervisors: Prof. Jun Han, Dr. Alan Colman, and Dr. Jian Yu

Committee members: Prof. Chengfei Liu and Dr. Rajesh Vasa
# 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 Adaptability ....................................................................... 3
   3.3 System-Context Relationships ....................................................................................... 3

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

5. Current Progress .................................................................................................................... 5
   
   5.1 Activities up to 12 Months Ago .................................................................................. 5
   5.2 Activities during the Last 12 Months ........................................................................... 5

6. Approach Validation .............................................................................................................. 5

7. Publications ............................................................................................................................ 5

8. Schedule ................................................................................................................................. 6

9. Thesis Structure ..................................................................................................................... 7

References .................................................................................................................................. 8
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. In these circumstances, a software system needs to change itself as necessary to continue achieving its new and existing goals [1-3].

We focus in this research on developing software systems that are able to cope with context and/or requirements changes, which we call context-aware adaptive software systems. The changes that need to be taken into account by a software system can be anticipated or unanticipated changes. First, the anticipated changes are known beforehand at design time, and the system needs to have a set of predefined reactions (i.e. system adaptations) to them when they happen. Second, the system is usually deployed into an environment that is not totally anticipated at the design time or new requirements need to be taken into account at runtime (i.e. unanticipated changes). To cope with these unknown changes, the system should have the ability to incorporate new context information, functionality, and corresponding system adaptations at runtime.

For a system to have the ability to cope with anticipated and unanticipated 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 behavior (i.e. runtime system adaptability) to keep achieving its new and existing goals; (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 behavior(s) into the overall system.

Existing research into systems that adapt themselves in response to context and/or requirements changes has been carried out largely in two research communities with their own foci: self-adaptive systems [1-3] and context-aware systems [4-6]. 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 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 a system in response to context and requirements 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 hardwiring the context processing and management operations together with the system functionality and its management. Furthermore, there is a very little consideration concerning unanticipated changes.

This research’s objective is to develop an approach to ease the development of software systems that are able to cope with anticipated and unanticipated changes 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. An example of such applications is the route planner. Below, we present a number of scenarios that motivates the need to develop a software system that has the ability to cope with anticipated and unanticipated changes.

Scenario 1: To plan a route for the driver, the route planner senses all available context information (e.g. route preference, traffic information, etc.) and selects a suitable route planning algorithm that takes into account this available information. After that, the route planner shows the available routes to the driver based on his destination and current context (e.g. his location, his route preferences, blocked roads, etc.). The driver selects a route and starts his journey. During the journey, the vehicle routes are re-calculated to take into account changes in the traffic information (e.g. congestion has formed en-route). Later, due to loss of communication with road side units, the system selects a route planner that does not take the traffic information into account.

Scenario 2: During the life time of the vehicle route planner 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. As such, the system may want to use such information for suggesting lane changes during the journey. In addition, the route planner provider may want to enhance the application functionality by adding an attraction finder service. This service finds attractions that surround a route selected by the driver.

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’s 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 en-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, at runtime, 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 needs to have a response to the anticipated changes, but also it needs to evolve itself to cope with the unanticipated changes. For example, the addition of the information about surrounding vehicles en-route 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. Similarly, in response to requirements changes, a service to find attractions needs to be added to the system and linked with some parts of it, so that the user can interact with this new service. As a consequence, the functional system should be able to evolve itself to cope with the unanticipated changes. In addition, to increase the users’ satisfaction, the route planning system should be available 24/7 (24 hours a day 7 days a week). As such, the changes to the route planning system to cope with the unanticipated changes need to be applied while it is in operation (i.e. without stopping and then restarting the system).

**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 from a system’s configuration and/or behavior 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. 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 relationship 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 so that their overhead on the system is reduced and the system can function properly with only the available context information [2, 7]. 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.

### 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 and requirements changes. In this section, we analyze existing approaches with regard to these requirements and identify the research gaps.

#### 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 [8-13]. When there are a large number of these context variables, the system’s 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 [14-17]. 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 [2, 7]. 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 [18-19]. These approaches only support 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’s state changes can also affect the context model structure. As such, an approach is needed to support changes to the context model structure at runtime through disabling the unwanted context entities to reduce the context monitoring overhead and re-enabling these entities when they are needed.
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 [4–5]. An approach has been introduced to enable the system management changes [20]. In this approach, the addition of a new adaptation rule may make the system 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 relationship between context model entities (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 high level context information). As a consequence, an approach is needed for the context model to make it able to incorporate new context information at runtime while considering the above aspects.

RQ1: How to (a) enable context model entities selection at runtime to reduce the context monitoring overhead and (b) make the context model incorporate new context information at runtime?

3.2 Functional System Runtime Adaptability

Runtime functional system changes: In response to context changes, the functional system needs to adapt itself [21]. 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 [22–30]. 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 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 elements and their 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 [8, 12, 17]. But, their models maintain only the system state (to initiate the adaptation process when system goals are violated) 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 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 [31–33]. 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 approach that supports the realizations of the unanticipated functional system changes at runtime.

RQ2: How to develop a model driven approach that supports the realizations of unanticipated functional system changes at runtime (i.e. runtime update to the functional system)?

3.3 System-Context Relationships

Explicit representation of the relationships: The system and its context are related entities where context changes affect the system’s operation and management. 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. Much of existing research models the system-context relationships implicitly [34]. 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 hardwire the context model with the relationships model. As such, the relationships’ modelling complexity is increased (e.g. [8, 10, 20]). 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 [20]. In their approach [20], 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 support runtime changes to the operational relationships (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 on the system (e.g. the removal of unwanted relationships), where their management relationships changes are performed manually. As such, an approach is needed that supports changes to the operational and management system-context relationships while the system is in operation.
RQ3: How to capture the system-context relationships and support their runtime adaptation to reduce their overhead and their runtime evolution to incorporate new relationships?

4. General Approach

The process that a software engineer needs to follow in developing and evolving context-aware adaptive systems is shown in Figure 1. This process is divided into two stages. The first stage is the system development phase (the top part of Figure 1). In that phase, the software engineer models a system that has the ability to cope with changes that are known beforehand (i.e. anticipated changes) and uses the designed models to implement the system. The second stage (system evolution phase) starts when there is a running instance of the system (the bottom part of Figure 1). During this stage, the system models (as designed) are changed to incorporate a set of elements to cope with unanticipated context and requirements changes. Then, the changes that are performed into the system model are reflected in the running system. In the following, we describe these steps and our approaches to support them in general.

Step 1: To model a context-aware adaptive software system, the system functional requirements need to be mapped to functional models (e.g. system’s structure model). In addition, context information that is required by the system to continue its operation or can cause a system adaptation is used to define the context model. Furthermore, an adaptation model is defined that specifies the system reactions to anticipated context changes. To support this step, we have introduced a modelling language that separates but relates the context model and system model, so that they and their relationships can be clearly captured [35-36].

Step 2: The designed models are used to guide the system implementation stage. First, the system functional models are used as a blueprint for implementing the system’s functionality while the context model defines what the needed context providers are (i.e. context acquisition mechanisms). In addition, to adapt the system in response to context changes at runtime, the adaptive behaviour model is transformed to code that manages the system while it is in operation (i.e. system’s management). Finally, the system management needs context information to decide a set of changes which in turn are performed on the system, and the system functionality also needs the context information to continue their operations. As such, a set of relationships are implemented to capture the different relationships between the system elements (i.e. functional, contextual, and management elements). To easy this implementation task, we have proposed a model-driven approach, where our modelling language is used to represent the system and then the system model is transformed to the system implementation automatically [37]. To enable this realization step, we have used the ROAD framework as the implementation platform [38].

Step 3: To incorporate new context information and the system reaction(s) to it, the context model need to be modified by adding the new context information, and then the required changes into the functional system and the adaptations models are performed accordingly. Similarly, to consider a new functional requirement, the system’s functional model is modified to include such functionality while the context model is changed to include context information that may be required by that functionality to operate correctly. Also, some changes are performed to the adaptive behaviour model, so that the system can adapt properly by taking the new added functionality into account. To enable the system model evolution, we have developed a tool that enables the software engineer to load a running system model. Then, he can change the system model to incorporate new requirements and/or context information. We have also proposed a set of patterns to assist the system model’s evolution. These patterns specify the changes needed by the system in response to context and/or requirements changes.

Step 4: The system needs to evolve while it is running, to take into account a new context and/or requirement. As such, the system should be developed with this feature in mind, where the required changes (as in the models of step 3) can be synchronized with the running system. To support this feature, we have engineered the developed system with the ability to add, remove, or modify its elements while it is in operation. At runtime, to identify the required changes that need to be reflected into the running system, a difference between the running system model and the changed model is computed. Then, we use this difference to generate adaptation actions that are used to evolve the system through executing the adaptation functions that are engineered into the system during its development time.
5. Current Progress

During the last two and half years, we were working on approaches that support the above process to developing and evolving context-aware adaptive systems.

5.1 Activities up to 12 Months Ago

During the first one and half year, four tasks have been completed. Firstly, we have performed an analytical survey of context-aware systems and self-adaptive systems to identify the research gaps [34]. Secondly, in order to structure our work, we have proposed an architectural approach for context-aware adaptive systems. This architecture serves as a roadmap for our research [36].

Thirdly, we have developed an approach for modeling and realizing context-aware adaptive systems [35]. This approach is based on a component model that explicitly supports in the component interfaces the definition of the required and provided functionalities, context information and management actions. Finally, we have investigated the system adaptive behavior validation 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].

5.2 Activities during the Last 12 Months

During the last year, three tasks have been completed. First, we were planning to start the work on system’s runtime evolution (i.e. Steps 3 and 4) at the beginning of this year. However, the approach that we developed for the system development phase was not flexible enough to support the runtime evolution. As such, we have enhanced the work we have done in modelling and realizing context-aware adaptive system though using the ROAD framework as a realization platform. In order to do that, we have developed a tool that enables the modelling of the system and automatic transformation of this model to a model that is compatible with the ROAD framework [40].

Secondly, we have developed a technique to enable the system runtime evolution. We have also developed a tool to manage the system at runtime. This tool can be used to load the model of a running system. Then, this model is changed by the software engineer to incorporate new context information and/or requirements. Then he can press a button that will initiate the automatic application of the required changes to the running system. We have also proposed a set of evolution patterns that can be used by the engineer to define what elements need to be changed in the system model to take into account new context information, new functional requirement, or new adaptation requirement.

Thirdly, our approaches for both system’s development and runtime evolution works at the system design level where the system requirements are assumed to be already captured and analyzed for design. To support the capture of context-aware adaptive systems requirements, we have extended the UML sequence diagram so that it can capture the context-awareness and adaptability aspects. Using this extended sequence diagram, the system requirements are represented as a set of scenarios. We also support the automatic transformation of these scenarios to an initial system’s design. This design can be completed by the software engineer, and then it can be transformed to an executable system automatically.

6. Approach Validation

To evaluate our approach, we will measure the engineering effort and the runtime adaptation overhead [41-43]. To calculate the engineering effort, we will measure the time and cost needed to develop a context-aware adaptive system using our approach and compare it with using traditional development/evolution approaches (i.e. to quantify the development time and cost reduction when the new approach is used). Then, to assure our approach’s applicability, we will measure the overhead caused by adding the adaptivity feature to a running system. Our approach’s overhead can be measured using the following three metrics. First, system’s working time vs. adaptivity time which is the CPU time spent by the system to decide and act the required system adaptation comparing the time spent to perform its core functionality. Second, the system response time variation in the presence of the system adaptivity comparing to the case where no adaptivity is introduced. Finally, we will measure the amount of runtime memory usage that is increased by adding the adaptivity to the software system.

7. Publications

Published and Accepted Papers:


Under Review:

Under Preparation:
• A paper reporting our organizational approach to developing context-aware adaptive systems will be submitted to the Joint 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture, Helsinki, Finland, August 20–24, 2012 (an initial draft can be found in [40]).
• Our analytical survey into context-aware adaptive systems will be submitted to the ACM Transactions on Autonomous and Adaptive Systems (an initial draft can be found in [34]).

Future Publications:
• We are planning to extend the paper submitted to the CBSE conference to include formal validation of the system’s functional and adaptive behavior and submit it to the IEEE Transactions on Software Engineering.
• In our approach to support runtime evolution, we support a manual update of the system model using a set of patterns and then these changes are reflected to the running system. We are planning to automate these patterns and possibly use artificial intelligence techniques to make the runtime evolution semi-automated (venue will be selected).

8. Schedule
The work during the PhD period is scheduled as in Table 1. During the first year, a literature review has been completed and then an analysis of existing approaches was presented as a technical report [34]. While we were doing the analysis, we have developed an architectural approach for context-aware adaptive systems [36]. 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 changes.

In the design time phase (around twelve months), we have developed an approach to modelling and realizing software systems that are able to cope with context changes that are known beforehand at design time. In this regard, we have developed (a) an approach to model the system, the context, and their relationships explicitly [35]; (b) a tool to generate the system implementations from their models [37]; (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, we have proposed an approach to enable system’s runtime evolution through changing the system model and reflecting these changes to the running system. We have also proposed a set of evolution patterns to ease that task. During the next year, we will automate these patterns so that the system can evolve semi-automatically with limited human involvement. In the mean time, we will start the thesis writing.

Table 1: PhD activities schedule and their allocated periods

<table>
<thead>
<tr>
<th>Activities</th>
<th>Allocated Time Periods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literature review</td>
<td>Aug’09 - Nov’09</td>
</tr>
<tr>
<td>Architecture for Context-aware Adaptive Systems</td>
<td>Dec’09 - Feb’10</td>
</tr>
<tr>
<td>Modelling and Realizing Context-aware Adaptive Systems</td>
<td>Mar’10 - May’10</td>
</tr>
<tr>
<td>Enabling the System Runtime Evolution</td>
<td>Jun’10 - Aug’10</td>
</tr>
<tr>
<td>Modelling Context-aware Adaptive Systems Requires</td>
<td>Sep’10 - Nov’10</td>
</tr>
<tr>
<td>Approach Validation and Refinement, Papers Writing</td>
<td>Dec’10 - Feb’11</td>
</tr>
<tr>
<td>Thesis writing</td>
<td>Mar’11 - May’11</td>
</tr>
<tr>
<td></td>
<td>Jun’11 - Aug’11</td>
</tr>
<tr>
<td></td>
<td>Sep’11 - Nov’11</td>
</tr>
<tr>
<td></td>
<td>Dec’11 - Feb’12</td>
</tr>
<tr>
<td></td>
<td>Mar’12 - May’12</td>
</tr>
<tr>
<td></td>
<td>Jun’12 - Aug’12</td>
</tr>
<tr>
<td></td>
<td>Sep’12 - Nov’12</td>
</tr>
<tr>
<td></td>
<td>Dec’12 - Feb’13</td>
</tr>
<tr>
<td></td>
<td>Mar’13 - May’13</td>
</tr>
</tbody>
</table>
9. Thesis Structure

The thesis is structured into three parts. The first part consists of an introduction to 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 changes. Finally, the third part contains our developed tool, the approach validation, and 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
   2.1.3 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-driven Development and Evolution of Context-aware Adaptive Software Systems

Chapter 4: Eliciting Context-aware Adaptive Systems Requirements
4.1 Extending the Sequence Diagram Meta-model to Capture Context-aware Adaptive Software System Requirements
4.2 The Modified Sequence Diagram Model
4.3 Illustrative Example
4.4 Summary

Chapter 5: Modelling and Realizing Context-aware Adaptive Systems
5.1 Modelling Context-aware Adaptive Software Systems
5.2 Realizing Context-aware Adaptive Systems using ROAD Framework
5.3 Illustrative Example
5.4 Summary

Chapter 6: Enabling the System Runtime Evolution
6.1 Evolution Patterns to cope with Unanticipated Context Changes
6.2 Evolution Patterns to add New Requirements
6.3 Evolution Patterns to incorporate New Adaptive Behaviors
6.4 Illustrative Example
6.5 Summary

Part III: Implemented Tool, Validation, and Conclusions

Chapter 7: Tool Support
7.1 Context-aware Adaptive Systems Development and Evolution Tool
7.2 The Architecture of the Tool and its Design
7.3 Tool Examples
7.4 Summary
Chapter 8: Case Studies
8.1 Case Study 1: Context-aware Route Planning System
8.2 Case Study 2: Context-aware Travel Guide System
8.3 Summary

Chapter 9: Empirical Validation
9.1 Engineering Effort
9.2 Runtime Adaptation Overhead
9.3 Summary

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

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

References