Scenario-driven Development of Context-aware Adaptive Software Systems

Technical Report: C3-516_07

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

February 2012
# Table of contents

## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  INTRODUCTION</td>
<td>2</td>
</tr>
<tr>
<td>2  MOTIVATING SCENARIOS AND REQUIREMENTS ANALYSIS</td>
<td>3</td>
</tr>
<tr>
<td>3  A PROCESS TO DEVELOPING CONTEXT-AWARE ADAPTIVE SOFTWARE SYSTEMS</td>
<td>4</td>
</tr>
<tr>
<td>4  CONTEXT-AWARE ADAPTIVE SOFTWARE SYSTEMS REQUIREMENTS ELICITATION</td>
<td>5</td>
</tr>
<tr>
<td>4.1 Extending the UML Sequence Diagram Meta-Model to capture Context-awareness and Adaptability Aspects</td>
<td>5</td>
</tr>
<tr>
<td>4.2 Elicitation of Context-aware Adaptive Systems Requirements</td>
<td>7</td>
</tr>
<tr>
<td>5  SYNTHESIS OF THE SYSTEM MODEL</td>
<td>9</td>
</tr>
<tr>
<td>5.1 Context-aware Adaptive Software Systems Modelling</td>
<td>9</td>
</tr>
<tr>
<td>5.2 Synthesis of a System Model from its Scenarios</td>
<td>11</td>
</tr>
<tr>
<td>5.2.1 Synthesis of the System Functional Composite</td>
<td>12</td>
</tr>
<tr>
<td>5.2.2 Synthesis of the System Management Composite</td>
<td>16</td>
</tr>
<tr>
<td>6  IMPLEMENTATION</td>
<td>18</td>
</tr>
<tr>
<td>7  RELATED WORK AND DISCUSSIONS</td>
<td>20</td>
</tr>
<tr>
<td>7.1 Requirements Elicitation</td>
<td>20</td>
</tr>
<tr>
<td>7.2 System’s Design and Construction</td>
<td>21</td>
</tr>
<tr>
<td>8  CONCLUSIONS AND FUTURE WORK</td>
<td>21</td>
</tr>
</tbody>
</table>
Faculty of Information and Communication Technologies
Swinburne University of Technology

Scenario-driven Development of Context-aware Adaptive Software Systems

Mahmoud Hussein, Jian Yu, Jun Han, and Alan Colman

Abstract

Context-awareness and adaptability are highly desirable features for software systems that operate in dynamic environments. In recent years, a number of approaches have been proposed to support the development of such systems. However, elicitation of a context-aware adaptive system requirements and synthesis of the system model from its requirements are still major challenges. In this paper, we propose a novel approach to scenario-driven development of context-aware adaptive systems. Our approach enables the elicitation of the system requirements as a set of scenarios. In particular, we differentiate functional scenarios from adaptation scenarios. The functional scenarios capture the system’s functionality while taking the context information into account to operate effectively. The adaptation scenarios represent the system’s runtime adaptation to cope with the context changes. We also support synthesis of the system model from its scenarios. This model is then completed by the software engineer to add elements that are related to the system’s solution space and cannot be synthesised from the scenarios directly. We have developed a tool that enables the system’s implementations generation from their models. A case study of developing a context-aware travel guide system is also presented to demonstrate the viability of our approach.

Keywords: Context-awareness, Self-adaptation, Requirements Elicitation, Systems Modelling, Scenario-driven Development.

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]. In these circumstances, a software system needs to change itself as necessary to continue achieving and/or preserving its goals. A challenge is how to specify, design, verify, and realize such systems that evolve at runtime [1-2].

We focus, in this paper, 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 systems. Recent surveys show that a number of approaches have been proposed to support the development of context-aware adaptive systems [1-4]. Generally, most of these approaches focus on supporting the design and construction of systems that have the ability to be adapted at runtime to cope with context changes with less attention to supporting the elicitation of the systems’ requirements. Thus, two major challenges still remain that hinder easy and effective development of such software systems. First, to elicit the requirements of a context-aware adaptive system, there is a need for an
approach that considers the system’s context and possible adaptation actions to cope with context changes during requirements elicitation. Also, this approach needs to follow an industry standard, so it can be easily adopted by software practitioners. Second, to ease the system development, an approach is needed that can be used to synthesis the system model from its requirements. Then, this model is transformed to a real system automatically. But, existing approaches do not consider these two challenges.

In this paper, we propose a novel scenario-driven approach for the development of context-aware adaptive systems. Our approach elicits the system’s requirements as two sets of scenarios: functional and adaptation scenarios. The functional scenarios capture the system’s functionality while taking the context information into account to operate effectively. The adaptation scenarios represent the system’s runtime adaptation to cope with context changes. In requirements elicitation, we have adopted a scenario-based approach because (1) its simplicity and intuitive graphical representation facilitate stakeholder involvement in capturing the system requirements; (2) it also has a well-understood and widely accepted semantics [5-6] (e.g. the UML sequence diagram [7]). As such, we extended the UML sequence diagram to capture the system’s context-awareness and adaptability aspects. We do so through having three types of participants: functional, contextual, and management participants that provide system’s functionality, provide context information, and decide adaptation actions to cope with context changes respectively. These participants also interact with each other through special types of messages based on the participants’ types. Our approach also supports synthesis of the system model from its scenarios’ descriptions. Later on, the synthesized model can be completed and/or modified by the software engineer to add elements that are related to the system’s solution space. To realize the system, we follow a model-driven approach [8], where we transform the system’s synthesised model to a real system. To do so, we have developed a tool that supports the system implementations generation from their models. Finally, we demonstrate our approach through the development of a context-aware travel guide system.

The remainder of the paper is organized as follows. We start by introducing a motivating scenario in Section 2. In Section 3, we present a process that the software engineer need to follows in developing context-aware adaptive systems. Our approach to eliciting the system requirements is presented in Section 4. The process to synthesis the system’s model from its scenarios is described in Section 5. Our tool support and how it can be used to develop the context-aware travel guide system are discussed in Section 6. Section 7 analyses existing work with respect to our approach. Finally, we conclude the paper in Section 8.

2 Motivating Scenarios and Requirements Analysis

The travel guide system helps a tourist to find attractions, plan his trip by providing suitable routes, and locate a restaurant. Below are some scenarios the tourist experiences in using this system during his one-day tour in Melbourne.

Scene 1: In the morning, the tourist starts to plan his trip. Based on his preferences (e.g. outdoor attractions) and the weather forecast for that day (e.g. sunny), the travel guide suggests to him a number of attractions such as Melbourne Aquarium, Royal Botanic Gardens, Melbourne Zoo, etc. He selected some of these attractions to visit using his rented car, and then a set of routes are displayed to him. These routes are calculated based on his current location, his attractions list, his driving preferences (e.g. shortest route), and current traffic information (e.g. congested roads). He selected a suitable route and started to explore the attractions.
**Scene 2:** At lunch time, the application suggested to him a number of nearby restaurants that matches his food preferences while taking into account the locations of the remaining attractions he plans to visit. When he selects a restaurant, the trip route is re-planned automatically to take into account the restaurant location.

To develop the travel guide system that meets the tourist needs, a set of general requirements need to be considered:

*Compose a context-aware system* (Req. 1): The travel guide system need to be consists of a set of functional services (e.g. route planner) that interact with each other to meet the tourist needs, while considering some quality requirements (e.g. fast route planner). In addition, these services should take into account the context information with a certain quality (if required) to give the tourist better suggestions. For example, the route planner needs the traffic information updated to the last minute to provide accurate estimations for the routes travel times.

*Adapt the system while it is in operation* (Req. 2): The travel guide provider may want to provide the attractions finder service free, while the tourist should pay to use the other services. As such, while the travel guide system is in operation, the tourist may want to include the route planner service that is not provided free to him initially. To include such service, several changes need to be applied into the running system. Firstly, the system needs to be adapted by incorporating the route planning service (i.e. *adding a functional service*). Secondly, to find a suitable route for the tourist, there is also a need to acquire the tourist driving preferences and current traffic information and use them in calculating and suggesting the routes (i.e. changing the context model by *including context providers* and their relationships with the system functionality). Finally, the traffic information may become unavailable for a period of time (due to communication failure with road side units, for example), and then the travel guide provider need to have two route planners. One of them considers the traffic information in calculating the routes while the other does not take it into account. To switch between these two route planners while the system is running, the system *adaptation logic* needs to be changed, so that a suitable route planner can be selected based on the traffic information availability.

## 3 A Process to Developing Context-aware Adaptive Software Systems

To develop a context-aware adaptive system such as the travel guide system described in Section 2, it needs to be composed of a set of functional services that interact with each other to provide the system’s functionality while taking the context information into account (i.e. system’s *context-awareness*). Also, while the system is in operation it needs to adapt itself in response to context changes to preserve the achievement of the user needs (i.e. system’s *adaptability*). In this paper, we propose a scenario-driven approach to developing context-aware adaptive systems. Figure 1 shows a process that a software development team needs to follow to develop a system using our approach. In the following, we describe these steps in general and our approaches to support these steps are discussed in next sections.

**Step 1:** The success of a software system is measured by the degree to which it meets the users’ requirements, so that capturing these requirements is an important step in developing the software system. It is also often that the users find it difficult to express their needs. As a consequence, the requirement engineer resort to capture information about the tasks that the users currently perform or might want to perform as a set of scenarios [9]. To enable elicitation of a context-aware adaptive system’s scenarios, we extended the UML sequence diagram [7], which is widely used in practice to represent the system’s context-awareness and adaptability aspects (Section 4).
Step 2: To ease the system development and maintain a causal connection between the system’s design and its requirements [10], in this step, we synthesis the system’s initial design from its scenarios (Section 5). This design can be further modified by the software engineer. When he becomes satisfied by the system model, he proceeds to the implementation phase.

Step 3: Our approach is a model-driven approach. As such, we not only synthesis the system model from its scenarios but also we enable the generation of the system implementations from their models (Section 6).

4 Context-aware Adaptive Software Systems Requirements Elicitation

The In this section, we describe the extensions we made to the UML sequence diagram [7] to make it able to capture the system’s context-awareness and adaptability aspects. Then, we describe how to use the extended sequence diagram to elicit the travel guide system requirements (Step 1).

4.1 Extending the UML Sequence Diagram Meta-Model to capture Context-awareness and Adaptability Aspects

A meta-model of the UML sequence diagram is shown in the top part of Figure 2\(^1\). This meta-model has the main elements of the sequence diagram. Each sequence consists of a set of lifelines, messages, and interaction fragments. The lifelines represent the sequence participants while the messages capture the interactions between the participants. In addition, the interaction fragments within the sequence specify (1) the order in which the interactions are going to be executed and (2) the constraints on these interactions.

Examples of the interaction fragments are combined fragment, state invariant, occurrence specification, and interaction use as shown in Figure 2 [7]. First, the combined fragment is used to group a set of interactions and define their relationships (e.g. sequence, alterative, etc.). Each

\(^1\) This is a simplified version of the UML sequence diagram meta-model described in the UML Superstructure specifications [Chapter 14: http://www.omg.org/spec/UML/2.3/Superstructure/PDF/].
combined fragment consists of an operator and operands. The operator defines the type of the combined fragment (e.g. alternative (alt), parallel (par), loop, etc.) while the operands specify the interactions that are being grouped by the combined fragment operator. In addition, each operand can have a constraint as a Boolean expression that specifies when this interaction can be executed.

Second, the state invariant is a runtime constraint on a lifeline, and the evaluation of this constraint to true means that the running sequence is a valid trace. Third, the occurrence specification is the intersection point between the lifeline and a message and it is used to define what events are sent or received by a lifeline. Finally, a sequence can refer to another sequence and then the interaction use is used to specify that a sequence has a reference to another (i.e. sequence reference).

To elicit the requirements of a context-aware adaptive system, we extended the sequence diagram concepts that are described above (see the bottom part of Figure 1) to consider requirements we have identified in Section 2. Firstly, in context-aware adaptive systems the participants can provide system’s functionality, provide context information, or decide the required adaptation to cope with context changes (i.e. manage the system). As such, we have three types of lifelines: functional, contextual, and management.

Secondly, we have different types of participants, and they also exchange special types of messages (i.e. functional, contextual, or management message as shown in Figure 2). The functional messages are formed between two functional participants, where one participant requests a function from another. When a participant needs to be aware of some context information, a contextual message is formed between this participant and a contextual participant that provide this information. This message can be requesting a context by a participant (i.e. get context) or a participant is notified automatically when the context changes (i.e. notify context). To adapt the system while it is in operation, we have a set of management messages that specifies what adaptation actions need to be performed at runtime. The possible adaptation actions are changes that can be applied to the sequence diagram elements (e.g. message, lifelines, etc.). An example of management messages is shown in Figure 2. These messages are used to adapt lifelines of a sequence diagram (i.e. add and remove a lifeline). Similarly, different changes can be specified for the other sequence diagram elements.

![Figure 2: An extended meta-model of the UML sequence diagram](image)

---

2 To represent the whole system execution, a main sequence can be defined that refers to the system simple sequences.
Thirdly, in eliciting the system requirements, there is a need to define some qualities. These qualities are about the system functionality (e.g. response time, reliability, etc.) or about the context information (e.g. accuracy, freshness, etc.). As such, we extended the sequence diagram state invariant to represent the context and functional qualities (see Figure 2).

Finally, to specify what to adapt in response to context changes, we added the adaptation operator. This operator group a set of a management messages (i.e. the adaptation actions) that adapt the system to cope with the context changes. In addition, we extended the interaction constraint to specify the context situation that a system needs to adapt itself in (i.e. context condition in Figure 2). We call this combined fragment as an adaptation fragment. In the next sub-section, we use these concepts to elicit the travel guide system requirements as a set of scenarios.

4.2 Elicitation of Context-aware Adaptive Systems Requirements

To simplify the system requirements elicitation, we have two types of scenarios: functional and adaptation scenarios. The functional scenarios are used to elicit a system functional requirements and context information needed by the system to continue its operations (i.e. Req. 1), while the adaptation scenarios specify adaptation actions need to be applied into a running system to cope with context changes (i.e. Req. 2).

Functional Scenarios: This type of scenarios is used to capture the system’s functionality while taking the context information into account to operate effectively. As such, it consists of a set of functional and contextual participants that interact with each other through functional and contextual messages. An example of a functional scenario is shown in Figure 3. This scenario represents the user interactions with the travel guide system to plan a route while taking the user driving preferences and traffic information into account.

In the route planning scenario, we have two functional participants: user and route planner. We also have three contextual participants (i.e. location, user information, and traffic information) as shown in Figure 3. When the sequence starts the user current location is notified automatically to the user through the application graphical user interface, and then the user requests routes calculation either by providing his attractions list (i.e. using PlanRoute1) or by providing his location and attractions list (i.e. using PlanRoute2). To calculate the routes using “PlanRoute1”, the route planner requests the user location from the global positioning system (GPS) device, so that the routes are planned from the user’s current location to his selected attractions. After that, the route planner acquires the user driving preferences and the traffic information (i.e. blocked and congested roads) to calculate the routes effectively, and then the routes are calculated and provided to the user. Finally the user can select a suitable route using the “SelectRoute” sequence. This is a simple sequence that enables the user to view the calculated routes details, and he can select a suitable route. In this scenario, we also have two quality requirements (see Figure 3): (1) the route planning function need to be performed in less than 5 seconds, and (2) the freshness of the traffic information should be up to the last minute.

In naming the sequence’s participants and messages, we give each one an identifier, so it can be easily referenced by the adaptation scenarios. The identifier is two letters that represents the element type and an auto number to differentiate between elements of the same type (see Figure 3 for abbreviations of each element type and their meanings).
Adaptation Scenarios: In response to context changes, the system needs to adapt itself to keep achieving its goals. As such, we introduce the adaptation scenarios to enable the elicitation of the system adaptation requirements. These scenarios describe how the system shall be managed at runtime to cope with the context changes.

An adaptation scenario usually has the following elements. First, it needs to have a participant that need to be managed. In our case, this participant is the functional system. Second, to decide the required adaptation in response to context change, the scenario need to have a management participant that is responsible for that task. Finally, the scenario need to have contextual participants that provide the context information that causes the system adaptation. An example of an adaptation scenario is shown in Figure 4. This scenario is used to manage the functional scenario specified in Figure 3 (i.e. FS1). It has a management lifeline (i.e. organizer), three contextual lifelines (i.e. GPS device, user information, and traffic information), and a functional lifeline which need to be managed (i.e. the travel guide system). This sequence starts by requesting the GPS device availability. When the device is not available (i.e. location availability is false), the route planning scenario is adapted by removing (1) the location participant (i.e. CL1) which in turn removes all the message that this participant is involved in; (2) the PlanRoute1 message, so that the user can only request route calculation by providing his location manually; (3) the combined fragment “A1” where it is not needed anymore. The reverse of these three adaptation actions are performed when the GPS device become available as shown in the ELSE part in Figure 4. Another adaptation fragment (T2) is shown in the bottom of Figure 4. It is used to adapt the route planning scenario based on the availability of traffic and user information, where their corresponding lifelines are added and removed based on their availability.
5 Synthesis of the system Model

In this section, we describe our approach to model context-aware adaptive systems, and how we synthesis the system model from its scenarios.

5.1 Context-aware Adaptive Software Systems Modelling

To design context-aware adaptive systems, we follow an organizational approach. In the following, we describe what the organizational approach is and how its concepts fit our requirements to design context-aware adaptive systems.

A system as an organization is a set of dynamic relationships between its roles to maintain the system viability in a changing environment [11]. In such view, the relationships are used to specify the system roles position descriptions. These position descriptions specify what tasks the system roles should do, while there are players who actually perform the tasks by playing these roles (i.e. Req. 1). In addition, in response to environment changes, the system manager changes the system roles, their players’ bindings, and their relationships to maintain the system viability (i.e. Req. 2). For example, a business organization is a collection of roles (e.g. public officer, secretary, etc.) that are related to each other through contracts [12]. These contracts define the permissible interactions between the organization roles and their mutual obligations (i.e. what tasks a role can request from others and how well these tasks should be performed). The organization roles are played by employees or outsourced to external organizations. In addition, to maintain the business viability in the face of business market changes, the business organizer (manager) may change the business structure by adding roles, hiring employees, etc. [13].
Similarly, we can see the travel guide system as an organization. In this organization, there are a set of functional (e.g. route planner) and contextual (e.g. traffic information) roles that interact with each other to provide the system functionally while taking the context information into account. These functional and contextual roles are played by functional services and context providers respectively (i.e. Req. 1.) In addition, there is a manager that adapts this organization in response to context changes to keep achieving the user needs (i.e. Req. 2). For example, in response to unavailability of the used route planner, another one is selected to keep achieving the user needs. Because of the above correspondences, we follow the organizational approach in designing context-aware systems.

Following the organizational approach, a meta-model for a context-aware adaptive system is shown in Figure 5. The system composition consists of two main elements: functional and management composites. Both composites have a set of roles that are related to each other through contracts and each role can be played by one or more players. Firstly, the functional composite represents the system functionality that takes the context information into account to operate effectively. As such, this composite contain a set of functional roles that interacts with each other through functional contracts as shown in Figure 5. The functional contracts captures the permissible interactions between two functional roles (i.e. functional interactions) and the required quality of service (e.g. response time) of an operation that is request by an interaction (i.e. functional quality). In addition, to make the system’s functionality context-aware, there are a set of contextual roles bound to context providers to make the context information available. Then, to use this information, we have contextual contracts that are formed between context roles and functional roles to specify what context information functional roles require in order to continue their operations (i.e. context attribute). Also, a role may specify a quality of the required context, and then each contract can have one or more context qualities as shown in Figure 5.

Furthermore, to define the service’s behaviour, a set of processes are specified. In designing these processes, we follow an event-based approach [14], where a process is designed as a set of tasks. Each task has a pre (events that enable the execution of a task) and post (events that are generated upon a task completion) conditions as a set of events. The process also has two events that specify the process’s start and end as shown in Figure 4. We adopt this approach because it represents a process as a set of loosely coupled tasks that are related with each other through events, so that the process can be easily adapted by changing the pre and post events of its tasks.

To simplify the Figure, we have removed associations between the system roles and contracts. But, they are described in the text.

---

3 To simplify the Figure, we have removed associations between the system roles and contracts. But, they are described in the text.
Secondly, the management composite specifies how the system adapts in response to context changes to keep achieving the user needs. In this composition, we have the following elements.

Firstly, we have three types of roles: functional, contextual, and management. The functional roles represent the functional composites that need to be managed while the contextual roles represent the context information that causes a system adaptation. The management role bound with its player is used to manage the system by deciding the required adaptation (management) actions in response to context changes. The management role-player has a set of adaptation rules. Each rule consists of events, conditions, and actions. The rule events are the context changes, while the conditions represent the context situations that need an adaptation. The actions are the adaptation actions need to be applied into the system to cope with the context changes. Secondly, to capture the composite roles' relationships, we have two types of contracts: management and contextual contracts. The management contract is formed between a management role and a functional role to capture the possible management actions that can be applied into the functional role. These management actions are inferred from the adaptation rules, where each decided adaptation action need to be applied into the functional system (e.g. add role and remove role as shown in Figure 5). On the other hand, the contextual contract is formed between a management role and a contextual role to capture the context information that triggers the system adaptation.

In the above, we have described the modelling concepts in general. In the following, we describe them in details with examples from the travel guide system with a process to the synthesis of a system model from its scenarios.

### 5.2 Synthesis of a System Model from its Scenarios

In order to synthesis a system model from its scenarios, we automatically map the system’s functional scenarios to a functional composite and the system’s adaptation scenarios to a management composite using the synthesis algorithm shown in Listing 1. In Listing 1, we only show the general mappings. In the following, we describe these mappings in details. We also introduced specific notations and textual representations that are more convenient to the software engineers in modelling the system compared to using general purpose modelling languages [15].

**Listing 1: An algorithm to synthesis the system model from its scenarios**

```java
SystemModel synthesisOfSystemModel(FunctionalScenarios FS, AdaptationScenarios AS){
    SystemModel SM = new SystemModel();
    FunctionalComposite FC = synthesisOfFunctionalComposite(FS);
    ManagementComposite MC = synthesisOfManagementComposite(AS);
    SM.setFunctionalComposite(FC); SM.setManagementComposite(MC);
    Return SM;
}

FunctionalComposite synthesisOfFunctionalComposite(FunctionalScenarios FS){
    FunctionalComposite FC = new FunctionalComposite();
    SynthesisOfCompositeRoles (FC, FS);
    SynthesisOfFunctionalContracts(FC, FS);
    Return FC;
}

ManagementComposite synthesisOfManagementComposite (AdaptationScenarios AS){
    ManagementComposite MC = new ManagementComposite();
    SynthesisOfCompositeRoles (MC, AS);
    SynthesisOfManagementContracts(MC, AS);
    SynthesisOfContextualContracts(MC, AS);
    Return MC;
}
```
5.2.1 Synthesis of the System Functional Composite

The functional composite contains two aspects: system’s functionality and the context required by this functionality. First, the system’s functionality is modelled as a set of functional roles that interact with each other through functional contracts. In addition, each role can be played by one or more functional players. Second, the context model is represented through a set of contextual contracts that are formed between context and functional roles to capture the system contextual requirements. In addition, there are a set of context providers to make this context information available. Below, we discuss how to synthesis these aspects.

(A) Roles and their Players. In our model, the functional roles represent the system’s functionality. Similarly, we have functional participants that are involved in providing or requesting a function in the scenarios. As such, we map each functional participant to a functional role as shown in Figure 6 (e.g. user and route planner roles). Each synthesised functional role need to have one or more functional players to provide its functionality at runtime. But, the scenarios do not give much information about how many players a role can have. As such, we only generate one player for each role and the software engineer can later define more players. For example, he can specify two route planning algorithms that can play the route planner role as shown in Figure 6. RoutePlanner1 considers the traffic information in calculating the routes while RoutePlanner2 do not take the traffic information into account. Similar to mapping the functional participants to functional roles, the contextual participants are also mapped to contextual roles (e.g. location and traffic information as shown in Figure 6). To make the context information available, there is a need for context providers that monitor the context information. These providers are part of the system solution space, and then they should be specified by the software engineer and bound to the synthesised roles. For example, a weather role bound with the weather service is responsible for providing current temperature and rain level to the attraction finder service, so that a correct suggestion is given to the user based on current weather conditions as shown in Figure 6. An algorithm to synthesis the functional composite’s roles is shown in Listing 2. This algorithm map each participant to a role bound to a player.

Listing 2: Synthesising the system roles from a set of scenarios

```
void SynthesisOfCompositeRoles(Composite C, Scenarios S)
{
    ForEach Scenario s in S
    {
        ForEach Participant P in s
        {
            Role R = new Role(P.getName(), P.getType());
            if (!C.exists(R))
            {
                C.addRole(R);
                Player PL = new (R.getName(), R.getType())
                C.addPlayer(PL);
                R.bind(PL);
            }
        }
    }
}
```

(C) Functional Contracts. The functional contracts are used to capture interactions, conversation clauses, and quality of service between the system’s functional roles. This corresponds to functional messages between functional participants with the combined fragments specified on them and their functional qualities. As such, we traverse the scenarios and if there are interactions between two functional participants, a functional contract is created between the functional roles correspond to these participants.
Figure 6: The travel guide system model

Listing 3. Synthesising the system functional contracts from functional scenarios

```
void SynthesisOfFunctionalContracts (FunctionalComposite FC, FunctionalScenarios FS){
    ForEach Scenario S in FS{
        Message lastMessage;
        ForEach Message M in S{
            //Functional Interaction
            FunctionalInteraction FI = new FunctionalInteraction(M);
            FunctionalContact FunC = FC.getFunctionalContract(M.getRoleA(), M.getRoleB());
            if(FunC == null){
                FunC = new FunctionalContract(M.getRoleA(), M.getRoleB());
                FC.addFunctionalContract(FunC);
            }
            FunC.addFunctionalInteraction(FI);
            //Functional Interaction Quality
            if(M.hasQuality()){
                ForEach Quality Q in M{
                    if(!FunC.exists(Q)){
                        FunC.addQuality(FI.getId(), Q.getAttribute(), Q.getOperator(), Q.getRequiredValue());
                    }
                }
                //Creating New Conversation Clause
                if(lastMessage != null){
                    FunC.addConversationClause(new ConversationClause(lastMessage.getId(), "LeadsTo", M.getId()));
                }
            }
            lastMessage = M;
        }
    }
}
```

An Example of a Synthesized Contract “FC2”

Functional Contract ID: FC2: User_RoutePalnner
Parties: Role A: User; Role B: RoutePlanner
Interaction Clauses:
1: {requestRoutes1 (AttractionsList), AtoB, void};
2: {requestRoutes2 (Location, AttractionsList), AtoB, void};
Functional Qualities:
q1: {ResponseTime, LessThan, 5 seconds}
Then, we synthesize the contract items as follows. First, each contract needs to have an identifier and then we create the identifier automatically by contacting “FC” which means a functional contract and the count of existing contracts plus one. For example, the contract “FC2” is formed between the user and route planner roles as shown in Figure 6. Second, for each functional message, a functional interaction is added to the created contract. An example of an interaction “i2” is shown in Listing 3. This interaction has (a) an identifier (i2) and a name (RequestRoutes2); (b) two input parameters (e.g. attractions list and current location); (c) a direction to specify who is responsible for providing the operation included in that interaction (i.e. “AtoB” which mean the route planner role is responsible for providing route calculation operation); (d) context parameters (e.g. congested and blocked roads); (e) a return type (e.g. void). Third, the system functional qualities (e.g. response time, availability, and reliability) are mapped as they are to the contracts. For example, the quality of service “q1” in Listing 3 specifies that the request route operations “i1” should not take more than 5 seconds in calculating the routes. This quality of routes calculation operation corresponds to the functional quality defined in Figure 3.

(D) Behaviour Processes: To synthesize the functional composite’s behaviour, each scenario is transformed to a process as follow. The first interaction in the scenario is used to define the start event of the process while the process’s end event is the completion of the scenario’s last interaction. For example, the event “weather requested” is the start event of the “plan a trip” process while the event “route selected” is the process end event as shown in Figure 5-B. Then, each interaction fragment in the scenario is transformed to a task or a set of tasks based on its type. If an interaction is a message or a sequence reference, it is transformed to a single task. The task pre-condition is its previous interaction’s completion and the post-condition of the task is an event that specifies its completion to enable the execution of other dependent tasks. For example, the “provide attractions” message is transformed to “provide attractions” task that has “attractiions found” as pre-condition and “attraction provided” as post condition as shown in Figure 5-B. On the other hand, if an interaction is a combined fragment, a set of tasks correspond to this fragment are generated. For example, the alterative fragment is transformed to two groups of tasks. The first group contains the tasks that are going to be executed when the fragment condition is true while the second group is executed when the fragment condition is false. An example of a transformed alterative fragment is “find attraction” fragment shown in Figure 5-B. In this fragment, the operation used to find the attractions depends on the weather information availability where FindAttractions2 is used when the weather information is available while FindAttractions1 is used otherwise. The algorithm to synthesis the behaviour processes is described in Listing 4.

Listing 4: Synthesising the system’s behaviour processes

```java
1: void synthesisProcessModel(Scenario SM, ProcessModel PM) {
2:     //FM: A Service’s Scenario       AM: The Service’s Process Model
3:     Event lastEvent = new Event();
4:     Boolean Start = true;
5:     ArrayList<InteractionFragment> fragments = SM.getInteractionFragments();
6:     ForEach InteractionFragment interaction in fragments {
7:         if(Start == true){
8:             lastEvent.setType("Start");
9:             lastEvent = AddFragment(interaction, PM, lastEvent); Start = false;
10:         } lastEvent = AddFragment(interaction, PM, lastEvent);
11:     lastEvent.setType("End");
12: }
13: Event AddFragment(InteractionFragment interaction, ProcessModel PM, Event lastEvent){
14:     Event fragmentLastEvent = new Event();
15:     String interactionType = interaction.getType();
16:     if((interactionType.equals("Message")) || (interactionType.equals("Reference"))){
17:         fragmentLastEvent = AddSimpleFragment(interaction, PM, lastEvent);
18:     else if (interactionType.equals("CombinedFragment")){
```
The contextual contract defines what context information required by a functional role and the quality of this required context (e.g. accuracy, freshness, etc.). It also forms the connection between functional and context roles. In the functional scenarios, the context information is captured as contextual messages. As such, we map a set of context messages between a functional role and the quality of this required context (e.g. accuracy, freshness, etc.). It also forms the connection between functional and context roles.

In Listing 4, we synthesised a contextual contract “CC6” which is used to specify that the route planner role needs to know congested and blocked roads with freshness up to the last minute, so that the route planner can calculate routes effectively. In synthesising the contextual contracts, context qualities are mapped as they are defined in the functional scenarios (e.g. freshness of the traffic information as shown in Figure 3 and Listing 5).

Listing 5: Synthesising the system contextual contracts

```java
void SynthesisOfContextualContracts (Composite C, Scenarios S){
    ForEach Scenario s in S{
        ForEach Message M in s{
            if(M.getType().equals("Contextual")){
                //Context Attributes
                ContextAttribute CA = new ContextAttribute(M);
                ContextualContract ConC = FC.getContextualContract(M.getRoleA(), M.getRoleB());
                if(ConC == null){
                    ConnC = new ContextualContract(M.getRoleA(), M.getRoleB());
                    ConC.addContextualContract(ConnC);
                }
                ConC.addContextAttribute(CA);
            }
        }
    }
}
```
### Context Attributes' Qualities

```java
// Context Attributes' Qualities
if (M.hasQuality()) {
    foreach (Quality Q in M) {
        if (! ConC.exists(Q)) {
            ConC.addQuality(CA.getId(), Q.getAttribute(),
                            Q.getOperator(), Q.getRequiredValue());
        }
    }
}
```

### An Example of a Synthesized Contract “CC6”

**Contextual Contract ID**: CC6: TrafficInformation_RoutePlanner  
**Parties**: Context Source: TrafficInformation; Context Consumer: RoutePlanner;  
**Context Attributes**:  
- a1: CongestedRoads;  
- a2: BlockedRoads;  
**Context Attributes Quality**:  
- q1: {a1, freshness, LessThan, 1 minute}

### 5.2.2 Synthesis of the System Management Composite

The management composite consists of functional roles, contextual roles, management roles, contextual contracts, and management contracts. In the following, we discuss how we synthesis the management roles and contracts, where the others can be synthesised as described above.

**A) Management Roles and their Players.** The management participates in the adaptation scenarios have the same purpose of the management roles in our model. As such, for each management participant in the adaptation scenarios, we add a management role in the synthesized system model. An example is the system organizer role that is added to the management composite shown in Figure 6 and it corresponds to the organizer participant in Figure 4.

To adapt the system in response to context changes, there is a need for a mechanism to decide when and what to adapt. To do so, we modelled the management role-player as a set of Event-Condition-Action rules [16]. We adopted the rule-based approach to capture the system adaptive behavior because of its expressiveness and availability of tool support. The adaptation rules decide when and what to adapt where the event and condition of a rule specify when to adapt, and the rule action defines what to adapt. The events that activate the adaptation rules are usually context changes where the system needs to adapt itself in response to these changes. The rule condition is used to specify the context situation that needs a system reaction(s). The rule action is a set of adaptation actions to cope with the context changes. In general, the adaptation actions are to add, remove, or modify a system element. For example, to change the system roles, we have three adaptation actions: add role, remove role, and change role-player binding. In the same manner, we have actions to add, remove, and change a functional, contextual, and management contracts.

To synthesis a management player, we map adaptation fragments in the adaptation scenarios to adaptation rules as shown in Listing 6. Each rule events are inferred from a fragment adaptation condition, where the events are the changes in the context attributes that are included into the adaptation condition. The rule condition is same as the adaptation condition. The rule actions are the adaptation actions that are corresponding to the management actions that are specified into the adaptation fragment. For example, the action `removeLifeline("x")` is transformed to `removeRole("x")`. Below is an example of an adaptation rule that is corresponding to the first adaptation fragment.
shown in Figure 4. This rule is activated (i.e. event) when the location information is not available (i.e. condition). In response to this change, the system adapts itself (i.e. actions) by (1) removing the contextual contracts “CC1” and “CC4”, and the “Location” role (these three actions correspond to removing the location life line); (2) modifying the functional contract “FC2” by removing the conversation clause “c1” and the interaction “i1” (these two actions are the result of transforming the management messages “MM2” and “MM3” in Figure 4).

Listing 6: Synthesising a management player

```java
void SynthesisOfManagementPlayer (ManagementComposite MC, AdaptationScenarios AS){
    ForEach Scenario S in AS{
        Player MP = MC.getManagementPlayer();
        ForEach CombinedFragment CF in S{
            AdaptationRule rule;
            if((CF.getType()).equals("Adaptation")){
                ContextCondition cond = CF.getCondition();
                ForEach ContextAttribute CA in cond{
                    rule.addContextChangEvent(CA);
                }
                rule.setCondition(cond);
                ForEach ManagementMessage MM in CF{
                    rule.addAction(MM);
                }
                MP.addAdaptationRule(rule);
            }
        }
    }
}
```

An example of an adaptation rule

Rule “AdaptationRule1”:
```
When ValueChanges (LocationAvailability);
if LocationAvailability == False;
do RemoveContract (“CC1”), RemoveContract (“CC4”),
RemoveRole(“Location”), RemoveInteraction(“FC2”,”i1”),
RemoveConversationClause (“FC2”, “c1”);
```

(B) Management Contract. To apply the adaptation actions, the functional composite that need to be managed should support the application of the adaptation actions. To specify the adaptation actions that need to be supported, we create a management contract “MC1” between the functional role and the management role. This contract defines what adaptation actions need to be performed into the functional system under the instruction of the management role as shown in Listing 7. In Listing 7, we also show part of the management contract “MC1”.

Listing 7. Synthesising the system management contracts

```java
void SynthesisOfManagementContracts (ManagementComposite MC, AdaptationScenarios AS){
    ForEach Scenario S in AS{
        ForEach Message M in S{
            if((M.getType()).equals("Management")){
                ManagementInteraction MI = new ManagementInteraction(M);
                ManagementContact ManC=MC.getManagementContract(M.getRoleA(), M.getRoleB());
                if(ManC == null){
                    ManC = new ManagementContract(M.getRoleA(), M.getRoleB());
                    MC.addManagementContract(ManC);
                }
                ManC.addFunctionalInteraction(MI);
            }
        }
    }
}
```
6 Implementation

To support the scenario-driven development of context-aware adaptive software systems, we have developed a tool. This tool enables the software engineer to represent a system’s functional and adaptation scenarios as shown in Figure 7-A (i.e. Step 1), synthesis of the system model from its scenarios automatically as discussed in the previous section (Figure 7-B is screenshots from the tool during the synthesis of the system model, i.e. Step 2), and generate the system implementations from their models. To support the system’s implementations generation (i.e. Step 3), we implemented a mechanism to support that. In the following, we describe this mechanism in details.

![Figure 7: Screenshots from the tool during the development of the travel guide system](image)

To realize context-aware adaptive systems (Step 3), we used ROAD framework where it follows the organizational approach as our approach does. This framework is an extension to the Apache Axis2\(^4\) to realize adaptive software [17]. To use this framework, we used our tool to transform the synthesized model described in the Section 5 to a model that is compatible with the ROAD framework. In the following, we describe some of the transformations we did (details about the transformations can be found in [18]).

Firstly, in ROAD model, the context information is maintained as a set of facts. Each fact contains one or more context attributes. These facts can be provided or consumed by the system roles. In our model, the contextual contracts are used to drive context roles descriptions, and then each context role can be seen as a collection of context attributes. This makes a correspondence between a fact in

\[^4\] http://axis.apache.org/axis2/java/core/
ROAD terms and context role derived from contextual contracts in our model. As such, we use the contextual contracts to drive context roles descriptions, and then we transform each role position description to a fact in ROAD model.

Secondly, to enable the execution of the adaptation rules of a management player, we transform them to Drools\textsuperscript{5} rules, so that the Drools rule engine can be used for their execution to decide the required adaptation actions while the system is in operation. In transforming the rules, we used the rule “\textit{When}” part to specify the rule event (e.g. the user need of the route planning service is changed). In addition, the rule “\textit{Then}” part is used for capturing both the rule condition and action. The adaptation rules are used to generate a script that contains adaptation actions need to be applied to a running system. This script is then used to adapt the running system by invoking the management interactions that correspond to the required adaptation actions. For example, removing the contextual contract “CC3” is transformed to \textit{organizer. removeContract(“CC3”).} The organizer variable is a reference to the running system organizer role.

The above transformation process is automated in our tool. When the software engineer completes the synthesized system model, he can press a button that generates the files required by the ROAD framework to deploy an instance of the system. To have a fully running system, we have developed a set of functional players and context providers. For example, we used Google maps services\textsuperscript{6} to develop the route planners’ players. We also have developed a graphical user interface (GUI) to enable the user interactions with the provided services (see Figure 8).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig8.png}
\caption{The context-aware adaptive travel guide system in action}
\end{figure}

In Figure 8-A, the system only includes the attraction finder service which is provided to the tourist free initially. This service suggested to the tourist a set of attractions based on his preference, his current location, and the weather forecast. He can select some of them to be included in his attractions list. To plan a route to see these attractions, the tourist requests the route planning service to be included in his system. As such, the system is adapted to include such service. After the internal adaptations are performed, the system GUI is also changed by including the route planning service. When this service becomes available, it acquires the tourist location, his attractions list, and current traffic information to suggest a suitable route (see Figure 8-B).

\textsuperscript{5} \url{http://www.jboss.org/drools}
\textsuperscript{6} \url{http://code.google.com/apis/maps/documentation/webservices/}
7 Related Work and Discussions

A number of approaches have been proposed to support the development of context-aware adaptive systems. These approaches support different phases of the system development. In this section, we discuss the approaches that are concerned with system’s requirements elicitation, and system’s design and construction (i.e. our approach focus).  

7.1 Requirements Elicitation

Recently, a number approaches has been proposed to support capturing context-aware adaptive systems’ requirements. In the following we describe some of them. RELAX is an approach to capturing adaptive system’s requirements through relaxing the system requirements to take into account context changes [19]. To do so, they first capture the system requirements (goals) as SHALL statements in a textual representation. Then, they classify these requirements as invariant (i.e. requirements that cannot be violated at runtime) or relax-able (i.e. requirements that are changeable at runtime to cope with the context changes) requirements. Finally, they introduce the RELAX operators (e.g. as early as possible, as close as possible to) to the relax-able requirements, so that they become flexible to cope with the runtime context changes. Another approach is proposed by Salifu et al. to capture the system’s monitoring and switching requirements [20]. The monitoring requirements are the context variables that cause a system adaptation, while the switching requirements specify the system switching between different states to keep achieving its goals in the faces of the context changes. To capture the context information and its effect into the system, they adopt the problem frames. Souza et al. proposed an approach to capture requirements that cause the system to adapt, which they call awareness-requirements (or AwReqs for short) [21]. To do so, they first capture the system requirements using a goal-oriented approach. Then, they define the awareness requirements as success or failure of the system requirements. An example of awareness requirement is “Communications networks working should have 99% success rate”.

The above approaches are based on capturing the system requirements as a set of goals. However, it is often that the users (stakeholders) find it difficult to articulate their needs (i.e. the system goals). As such, it becomes difficult to use these approaches to elicit context-aware adaptive systems’ requirements. To cope with this difficulty, the requirement engineer resort to capture information about the tasks that the users currently perform or might want to perform as a set of scenarios [9]. To support the requirement engineer in eliciting a context-aware adaptive system’s requirements, we proposed a scenario-based approach (see Section 4). As such, it can be easily used by the requirement engineer to capture the users’ needs. In addition, we adopted the UML sequence diagram which it widely accepted in practice. Similar to our approach is Adapt Cases approach [22]. This approach is used to model the system adaptivity at the logical design phase, so that the gap between the system’s analysis and design is filled. However, in their approach, the system requirements should be first captured using a goal-based approach (e.g. RELAX [19]). Then, the system goals are refined manually into one or more concrete logical UML uses case. As such, it has the same drawback of the above approaches (i.e. the difficulty in articulating the system goals). In our approach, we directly elicit the system requirements as a set of scenarios and we automatically

---

To support the system behaviour validation, we are planning to use the approach proposed by Uchitel et al.  

synthesize the system model from its scenarios. Also, they are more concerned with the system’s adaptability with less attention to the system’s context-awareness aspect.

7.2 System’s Design and Construction

In recent years, a number of approaches have been proposed to enable design and construction of systems that are able to adapt themselves to cope with context changes which know as adaptive systems. In the following, we describe some of them. Rainbow framework provides mechanisms to monitoring the context, performing the analysis of the context to initiate the adaptation process, selecting the required adaptation strategy, and effecting the needed changes to a running system [23]. To capture the system’s reaction(s) to context changes, they used a language called Stitch. An approach was introduced by Zhang and Cheng to creating formal models for a system’s behavior [24]. In their approach, the system’s adaptive behavior is separated from the non-adaptive behavior. This separation makes the system models easier to specify and verify. They use Petri-Nets to capture system’s adaptive behavior, where they use context change as guidance for the transition between the system states. MUSIC project is a component-based framework that is used to optimize a system overall utility in response to environment changes [25]. They have a quality of service (QoS) model, which describes the system composition together with the relevant QoS dimensions and how they are affected when the system is going to change from one configuration to another. The quality of service model is used for selecting a new configuration that has the best utility and is able to cope with context changes. Heaven et al. have developed an approach to adapt system’s structure in response to context changes while preserving its high level goals [26]. They use Labeled Transition Systems (LTS) to model the system domain. This model captures the states the system and its environment can be in, and the context changes are the actions that move the system from a state to another. Andrade et al. proposed an approach to cope with unanticipated changes to the adaptive behaviour of a system [27]. To do so, they separate the system adaptation logic from the system core artifacts (i.e. system’s functionality). They also represent the adaptation logic as a set of condition-action rules that specify the required adaptation actions in response to context changes. Each rule consists of two parts: (1) adaptation condition as an expression based on context attributes; (2) adaptation actions that represent the required changes to the system. These rules are constructed as a component-based system that can be changed at runtime. Morin et al. have proposed a technique to handle the exponential growth of the number of configurations that are derived from the system variability [28]. They combine model driven and aspect oriented approaches to cope with the complexity of adaptive systems.

In the above approaches, the system designer is responsible for translating the system requirements to its design manually. As such, this process is error-prone in large systems and the casual connections between the system requirements and its design is maintain manually which is a complex task. Also, these approaches lack the capability to elicit the system requirements. To tackle these challenges, our approach automatically synthesizes the system initial design from its scenarios, so that the casual connection between the system design and its requirements is maintained and the system design effort is also reduced. In addition, our approach separates the system model from the context model, so that we are able to adapt the context model at runtime while existing techniques assumes the context model is fixed and do not requires changes at runtime.

8 Conclusions and Future Work

In this paper, we have proposed a scenario-driven approach to ease the task of developing context-aware adaptive systems. To do so, we have extended the UML sequence diagram to capture the
system’s context-awareness and adaptability aspects. We also support the synthesis of the system model from its scenarios’ descriptions. Later on, the synthesized system model can be completed by the software engineer to add elements that are related to the system’s solution space. We also have developed a tool that supports the generation of the system implementations from their models. Finally, we demonstrate our approach through the development of a context-aware travel guide system.

Compared to existing approaches, our approach has the following key contributions. First, our approach enables elicitation of system requirements as two sets of scenarios: functional and adaptation scenarios. The functional scenarios capture the system’s functionality while taking the context information into account to operate effectively. The adaptation scenarios represent the system’s runtime adaptation to cope with the context changes. Second, we support synthesis of the system model from its scenarios’ descriptions. Finally, our tool enables scenario-driven development of context-aware adaptive systems.

As future work, our approach can be enhanced in several directions. Firstly, we will extend our approach to enable the system’s requirements evolution through defining new scenarios or modifying existing ones. Then, we synthesize the new system model from these scenarios and reflect the required changes to the running system automatically. Secondly, we have applied our approach to the travel guide case study and the results were promising. We will perform more validations to assess our approach’s applicability.

References


