Adv Syst Sci Appl 2018; **02**; 107-120
Published online at http://ijassa.ipu.ru/index.php/ijassa/article/view/539

An Approach for Developing Context-aware Adaptive Information Systems

Mahmoud Hussein[1]

[1)] *Faculty of Computers and Information, Menoufia University, Egypt*

E-mail: mahmoud.hussein@ci.menofia.edu.eg

Abstract. *Context-awareness* and *adaptability* are highly desirable features for modern information systems that are operating in dynamic environments. Unfortunately, such information systems are still difficult to build. Issues like (a) lack of an *approach to compose* a system from a set of functional services and context providers and (b) lack of a *mechanism to enable runtime adaptation* of the system in response to changes in its operating context need to be solved for easy and effective development of such information systems. In this paper, we introduce a novel approach to developing context-aware adaptive information systems. In composing the system, our approach explicitly separates but relates the system model and the context model, so that they and their relationships and changes can be clearly captured and managed. We also support runtime changes to the system, the context model, and their adaptation logic in response to the context changes. We do so by maintaining runtime representations of their models, and then we use these representations to realize the required changes. We have developed a tool to enable modelling the system and generate its implementations from their models. To demonstrate the viability of our approach, we used it to develop a context-aware adaptive travel guide system. We also assessed the performance of our approach through measuring the overhead caused by performing the system adaptation at runtime. The results demonstrate that our approach is capable of performing runtime adaptation with a small overhead.

Keywords: Information Systems Engineering, Context-awareness, System Adaptation, Context and System Modelling, Model Driven Development.

1. INTRODUCTION

There is an increasing demand for information systems that are aware of their contexts and dynamically adapt themselves at runtime in response to changes in these contexts [19]. Consider, for example, a context-aware travel guide system that a tourist can use to plan his trip to visit some attractions. This system needs to be composed of a set of functional services (e.g. route planner and attraction finder) that takes the context information (e.g. weather and traffic information) into account to operate effectively (e.g. give the tourist better suggestions for *routes* and *attractions*). Existing approaches focus on composing the system's functionality (e.g. [4, 6-8, 10, 14, 25-26, 29]), but they leave the task of capturing the context and relating it with the system to the system developers. As such, the system implementation stage become more complex and requires a developer with experience in building such type of systems. In practice this expertise is usually lacking, and it will be a tedious task for the developers.
While the travel guide system is in operation, it needs to adapt itself to keep achieving the tourist needs. For example, when the tourist wants to include a route planning service that was not provided to him free initially, the system should adapt itself by adding such service (i.e. changing *system's functionality*), the *context providers* needed by that service to operate effectively such as a traffic information provider (i.e. adapting the *context model*), and some *adaptation logic* to enable the switching between different route planning services based on

their availability and provided qualities (i.e. changing the *adaptation logic* itself). Most of existing approaches supports system's functionality changes. However, existing approaches intertwined the context with the system as discussed above. As such, the addition of a context provider and relating it to the system becomes a difficult and error prone task. In addition, they use goal-policy [14, 29], utility-policy [7-8, 26], or action-policy [4, 6, 10] to capture the system's *adaptation logic*. Most of these approaches are not able to change the adaptation logic at runtime except the approach proposed by Andrade et al. [2] that has the ability to change the adaptation logic manually at runtime by the system developer. However, the communications between the system developer and the running system are usually infrequent and sometimes impossible [18]. As such, the process of changing the system adaptation logic needs to be done automatically without the developer involvement.

To address the above drawbacks, in this paper, we propose a model-driven approach to developing *information systems* that are able to dynamically *adapt* themselves in response to changes in their contexts, which we call *context-aware adaptive systems*. Our approach has the following novel features. Firstly, in composing the system, we explicitly *separate but relate* the system model and the context model, so that their relationships, changes, and changes impact across the system and its contexts can be clearly captured and managed. Secondly, our approach enables the *runtime changes to the system, its contexts, and their adaptation logic*. We do so by maintaining runtime representations of their models and having two sets of rules: adaptation rules and adaptation meta-rules and. In response to context changes, the *adaptation rules* are used to decide the required changes to system and its contexts while the *adaptation meta-rules* specify changes that need to be applied into the adaptation rules themselves. In addition, our runtime environment enables applying the required adaptation actions. Finally, our tool can be used to model a context-aware adaptive system and generate its implementations from their models.

The remainder of the paper is organized as follows. We start by introducing a motivating scenario in section two. Our approach to *modelling* context-aware adaptive information systems is described in section three. In section four, we discuss our tool support for modelling and *realizing* context-aware adaptive systems. In this section, we also measured the runtime overhead of adding the context-awareness and adaptability features to an information system to assure our approach's applicability. Section five analyzes existing work with respect to our approach. Finally, we conclude the paper in section six.

2. MOTIVATING SCENARIO AND REQUIREMENTS ANALYSIS

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

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. blocked and 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 is planning 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 application need to be consisted of a set of *functional services* (i.e. attractions finder, restaurant locator, and 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 possible routes travel times and to display the routes that are less congested first. As such, there is a need for an approach that can be used to *compose* a set of *functional services* and *context providers* to provide the required context-aware functionalities while considering their quality requirements.

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 wants 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 planner 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 preference 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. THE APPROACH

To support the development of context-aware adaptive systems, we propose a *model-driven* approach. Model-driven development is the notion of constructing a model of the system that can be transformed to a real system [9]. In this section, we introduce an organisational approach and associated notation that can be used to *model* such systems and in the next section we discuss how to *transform* a system model created in this notation to a real system.

3.1. Composing Context-aware Adaptive Systems: An Organizational Approach

To develop a context-aware adaptive information system such as the travel guide system, it need to be composed of a set of functional services and context providers that interact with (related to) each other to meet the user needs (Req. 1). In addition, while the system is in operation it needs to adapt itself in response to context changes to preserve the achievement of the user goals (Req. 2).

A system as an organisation is "a set of *dynamic relationships* between its *roles* to maintain the system *viability* in a *changing environment"* [21]. The *relationships* can be seen as defining the system, where they are used to specify the system roles position descriptions. These descriptions specify what tasks the system roles should do, while there are *players* who actually perform the tasks by playing these roles. In addition, in response to environment changes, the system manager may change the system roles, their relationships, and their players' bindings to maintain the system viability. For example, a business organization is a

collection of *roles* (e.g. public officer, secretary, etc.) that are related to each other through *contracts* (relationships) [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). In addition, the organization roles are *played* by employees or outsourced to external organizations. Furthermore, to maintain the business *viability* in response to business *market changes*, the business manager may add roles, hire employees, etc [24].

Similarly, we can see the travel guide system as an organization. In this organization, there are a set of functional and contextual roles that interact with each other to provide the system required *functionally* while taking the *context* information into account. In addition, these functional and contextual roles are played by functional services and context providers respectively (i.e. *Req. 1*). Furthermore, there is an organizer role that is able to *adapt* this organization in response to context changes to keep achieving the user needs. For example, add the route planning service and the other elements related to it when the tourist needs such service (i.e. *Req 2*). Because of the above correspondences, we followed the *organizational approach* in modelling context-aware adaptive information systems.

Following the organizational approach, a meta-model for a context-aware adaptive system is shown in Figure 1. The system composition is consisted of a set of *roles* that are related to each other through *contracts* and each role can be played by one or more *players*. We have three types of roles: *functional*, *context*, or *organizer* as shown in Figure 1. The functional roles represent the systems functionality while context roles capture the context model. The organizer role bound to its player is used to manage the system by adapting it in response to context changes. In addition, to capture the system elements' different relationships, we have two types of contracts: *functional* and *contextual*. The functional contract is formed between two functional roles (i.e. role A and role B) to capture their functional interactions and mutual obligations. On the other hand, the contextual contract is formed between a context source and a context consumer to capture the contextual requirements and their required quality. Furthermore, for each type of role, we have a corresponding player (i.e. *context provider*, *organizer player*, *and functional player*). In the following, we describe these concepts in details with examples from the travel guide system.

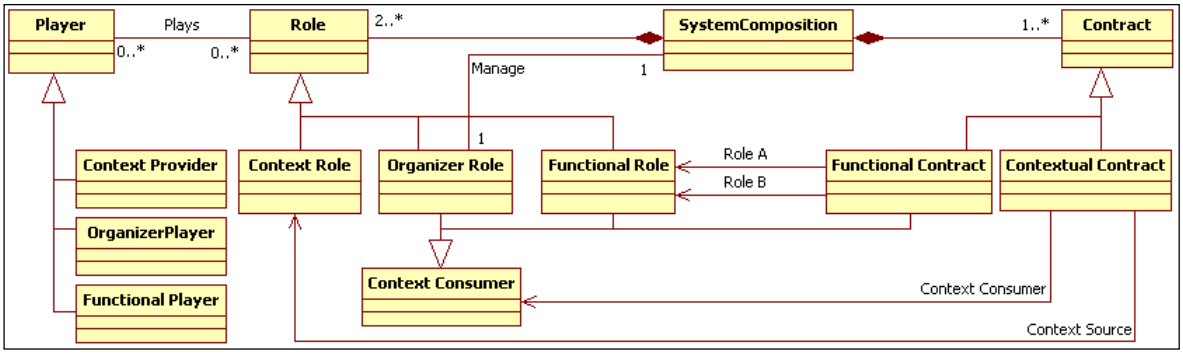

Fig. 1. A meta-model for context-aware adaptive software systems

3.2 Modelling System's Functionality and Context Model

Using the above meta-model concepts, we introduced a modelling language that can be used by the software engineer to model a context-aware adaptive system. The basic elements of this language are shown in Figure 2 and Listings 1 and 2. We introduced these specific graphical notations and textual representation to ease the system modelling task compared to using general purpose languages (e.g. UML) and to enable the system code generation from its model [17, 23]. A UML profile was an option for defining our domain-specific language [11], however the contracts in our model are more complex (see Listings 1 and 2) and it cannot

be captured easily using the UML profile concepts. In the following, we describe our modelling language elements and use them to model the travel guide system.

The Functional System Model: The system's functionality is modelled as a set of *functional roles* that interact with each other though *functional contracts*. In addition, each role can be played by one or more functional players.

Functional Contracts: The *functional contracts* are used to capture the relationships between the system functional elements and they have the following items. First, each contract has an *identifier* and it is formed between *two functional roles*. For example, the contract "*FC4*" is formed between the user and route planner roles as shown in Figure 2. Second, it has a set of permissible interactions between the contracted roles. Each interaction as shown in Listing 1 has (a) an identifier (e.g. *i2*) and a name (e.g. *request routes*), (b) zero or more input parameters (e.g. *destination and current location*), (c) a direction to specify who is responsible for providing the operation included in that interaction (e.g. "*AtoB*" which mean the route planer role is responsible for providing route calculation operation), (d) zero or more context parameters (e.g. *blocked and congested roads*), and (e) a return type (e.g. *routes*).

Third, the contract defines a set of conversion clauses that specifies the acceptable sequences of interactions between the functional roles. We used Interaction Rule Specification (IRS) language to specify these temporal constraints as shown in Listing 1 [15]. For example, "*c1*" specifies that "*request routes*" operation must be invoked before invoking "*select a route*" operation. Forth, to define the system non-functional requirements (e.g. response time, availability, and reliability), we specify a set of obligations on the functional interactions and we followed the Web Service Level Agreement (WSLA) language in defining these obligations [20]. For example, the obligation "*o1*" in Listing 1 specifies that the request route operation "*i1*" should not take more than 5 seconds in calculating the routes. Finally, the interactions between the functional roles can be permitted or blocked based on the context information. For example, the interaction "*i2*" cannot be performed when the traffic information is not available and then it needs to be blocked. To do so, each contract has a set of regulation rules to regulate its interactions. We used Event-Condition-Action rules to decide *permitting* or *blocking* an interaction [22]. An example of such rules is the rule shown in Listing 1. This rule is triggered when the interaction "*i2*" is requested (i.e. *i2.activated*). When the traffic information is not available, the interaction "*i2*" is blocked (*R2*) while it is permitted otherwise.

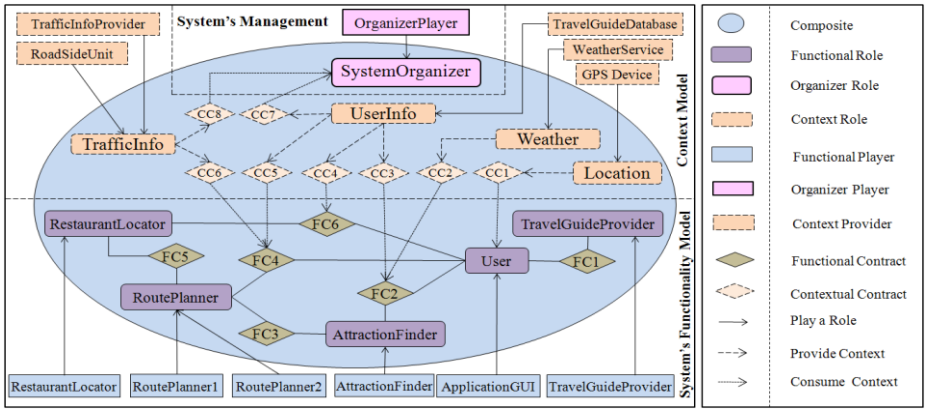

Fig. 2. The context-aware travel guide system model

Listing. 1. The functional contract between the user and the route planner functional roles

```
Functional Contract ID   FC4: User_RoutePalnner
Parties:                 Role A: User;        Role B: RoutePlanner;
Interaction Clauses:
    i1: {requestRoutes (Destination, CurrentLocation), AtoB, Routes};
    i2: {requestRoutes1 (Destination, CurrentLocation), ContextParameter
        (BlockedRoads, CongestedRoads), AtoB, Routes};
    i3: {selectRoutes (RouteID), AtoB};
Conversion Clauses (Temporal Constraints):
  c1: {i1 precedes i3 globally}          c2: {i2 precedes i3 globally}
Obligations:
  o1: {i1, ResponseTime, LessThan , 5 seconds}
  o2: {i2, ResponseTime, LessThan , 7 seconds}
Regulation Rules:
 R1:  {Block the interaction i2:
        When    i2.activated;
        if      TrafficInformationAvailability == False;
        do      i2.block( )};
```

Functional Roles and their Players: The *functional roles* represent the system's functionality where each role position description is derived from its relationships (contracts) with other functional roles (i.e. an abstract definition of the tasks that need to be performed by a player who will play that role). The functional role also has one or more *functional players* to provide its actual functionality at runtime. For example, there are two route planning algorithms that can play the *route planner role* as shown in Figure 2. *RoutePlanner1* considers the traffic information in calculating the routes while *RoutePlanner2* do not take the traffic information into account.

***The Context Model*:** The context model is represented through a set of contextual contracts that are formed between context sources and context consumers to capture the system contextual requirements. In addition, there are a set of context providers to make the context information available to the system.

Contextual Contracts: The contextual contract defines what context information required by a system element (i.e. the *context consumer* in Figure 1 such as functional contract, organizer role, or functional role) and the quality of this required context (e.g. accuracy, freshness, etc.). For example, in Listing 2, a contextual contract "*CC6*" is used to specify that the functional contract four needs to know (a) the congested roads with freshness up to the last minute and accuracy greater than 80%, and (b) the blocked roads with accuracy greater than 95% so that the route planner can calculate the routes effectively. In addition, it needs to know the traffic information availability to decide permitting or blocking the interaction "*i2*".

Listing. 2. The contextual contract between the traffic information role and the contract FC4

```
Contextual Contract ID   CC6: TrafficInformation_FC4
Parties: Context Source:TrafficInformation;
Context Consumer: FC4;
Context Attributes:
a1: CongestedRoads;
a2: BlockedRoads;
a3: TrafficInformationAvailability;
Context Attributes Quality:
    q1: {a1, freshness, LessThan , 1 minute}
    q2: {a1, accuracy, GreaterThan, 80%}
    q3: {a2, accuracy, GreaterThan, 95%}
```

Context Roles and Context Providers: The system contextual requirements are captured through a set of contextual contracts as discussed above. These contracts are then used to drive the *context roles* position description, where each context role bound with a *context provider* (each context role can have one or more context providers) is responsible for providing some context information. For example, a *weather* role bound to its provider (i.e. 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.

3.3 Engineering System's Adaptability through Organizer Role and its Player

To make the system able to adapt itself in response to context changes, there is a need for a mechanism to first decide *when and what to adapt* and then *apply* the decided adaptation actions. We do so through the system organizer role and its player.

We modelled the organizer player as a set of Event-Condition-Action rules that can be used to decide *when* and *what* to adapt [22]. The event and condition part of a rule specifies *when* to adapt, and the action part of the rule 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 functional and context 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 contract* and a *contextual contract*. To *apply* the adaptation actions, the organizer role is engineered with a set of *standard methods* that are corresponds to the adaptation actions so that the organizer player can invoke these methods to adapt the system while it is in operation.

An example of an adaptation rule is shown blow. This rule is activated when the tourist wants to include the route planning service in his application (i.e. *event*). In response to the changes in the tourist needs (i.e. he wants the route planning service, *condition*), the application adapts itself (i.e. *actions*) by adding a set of *roles* (e.g. route planner), adding a set of *contracts* (e.g. FC3 and FC4), and *bind players* with the added roles (e.g. bind route planner role with route planner one).

```
Rule "AdaptationRule1":  {
    When  ValueChanges (RoutePlannerSelected);
    if    RoutePlannerSelected == True;
    do    AddRole("RoutePlanner"), AddContract ("FC3"), AddContract ("FC4"),
          AddRole ("TrafficInformation"), AddContract ("CC6"), AddContract ("CC8"),
          Bind("RoutePlanner", "RoutePlanner1"), Bind("TrafficInformation",
          "RoadSideUnit")};
```

When the running system is changed, its adaptation rules may need to be adapted too. For example, after performing the adaptation actions in the above rule, we can see that the *route planner* role has two players and the *traffic information* role has two context providers as shown in Figure 2. As such, there is a need for some rules to decide the switching between the added roles-players. To do so, we specified another set of rules (i.e. adaptation *meta-rules*) that adapt the adaptation rules themselves. These rules have the same structure of the adaptation rules describe above, but they have different adaptation actions (i.e. *add rule* and *remove rule)*. An example of such meta-rules is show blow. In this rule, four adaptation rules are added to enable a correct selection of a route planner algorithm and a traffic information provider when the route planning service is included into the running system.

```
Rule "AdaptationMetaRule1":  {
    When  ValueChanges (RoutePlannerSelected);
    if    RoutePlannerSelected == True;
    do    AddRule("SelectRoutePlanner1"),   AddRule("SelectRoutePlanner2")};
          AddRule("SelectRoadSideUnit"),   AddRule("SelectTrafficInfoProvider")};
```

4. IMPLEMENTATION AND APPROACH EVALUATION

In this section, we describe the tool that has been developed to support the *modelling* and *realization* of context-aware adaptive information systems, and how to use this tool to develop

the travel guide system described in section two. We also measure the performance overhead of adding the adaptability feature to a software system using our approach to assure its applicability.

4.1 Tool Support for Modelling Context-aware Adaptive System

We have developed a tool to enable the modelling of a context-aware adaptive system. It enables the software engineer to specify the system *roles, players, contracts, and adaptation rules*. Screenshots from our tool during the system modelling are shown in Figure 3.

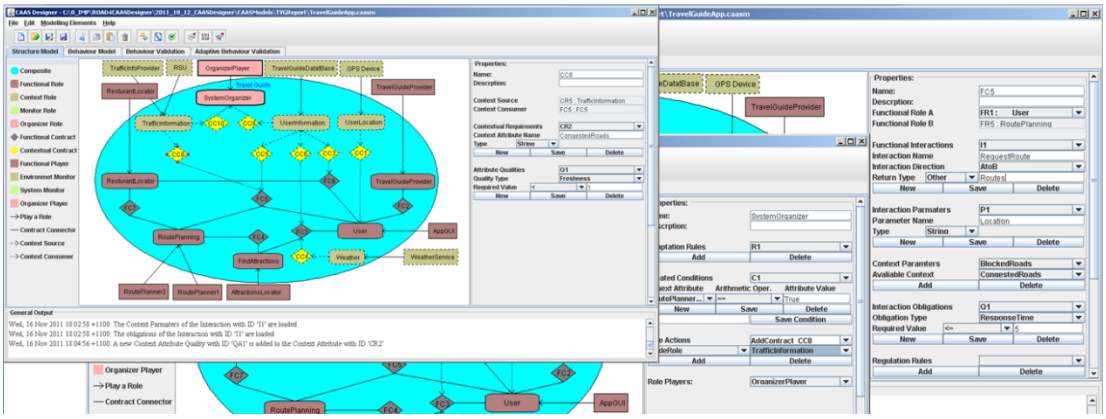

Fig. 3. Screenshots from our tool during the modelling of the travel guide application

To simplify the process of specifying the adaptation rules, we provide a GUI that helps the engineer in codifying these rules. This GUI is used to specify the rule *conditions* and *adaptation* actions. The rule events are directly inferred from the rule conditions, where they usually are the changes in the context attributes that are used in the rule conditions. An example is shown in Figure 4, where the engineer can specify the current situation i.e. the user wants to include the route planning service and a set of adaptation actions need to be performed in this situation such as *add route planner role* (Figure 4-A), *bind the route planner one player to route planner role* (Figure 4-B), and *add functional contract "FC4"* (Figure 4-C).

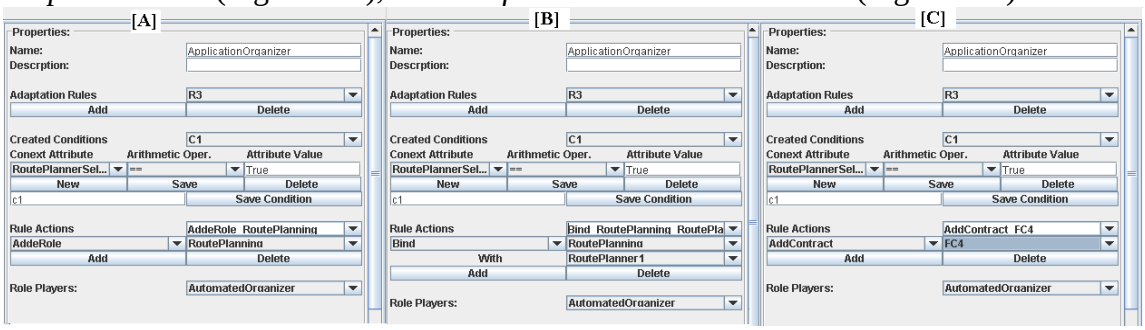

Fig. 4. Specifying the travel guide system adaptation rules using our tool

4.2 Realizing Context-aware Adaptive Systems

To realize context-aware adaptive systems, we used ROAD framework[1] where it follows the organizational approach as our approach does. This framework is an extension to the Apache Axis2[2] to realize adaptive software systems [16]. To use this framework, we used our tool to transform the model described in the previous section to a model that is compatible with the ROAD framework. In the following we describe the major transformations we did while the others are one-to-one mapping.

[1] http://www.swinburne.edu.au/ict/research/cs3/road/
[2] http://axis.apache.org/axis2/java/core/

First, the use of context information as extra parameter in a functional interaction is not considered in ROAD model. But, during the system execution, the functional contracts are used to mediate the interactions between the system roles. When an interaction reaches a contract a set of rules are executed to decide permitting or blocking it (in ROAD these rules are codified as Drools rules[3]). As such, we added a rule to these rules that is activated when a contextualized interaction is received. This rule is responsible for updating the context information required by this interaction before sending it to the destination role. An example of such rule is shown blow. This rule updates the context information (i.e. *CongestedRoads* and *BlockedRoads*) of the request route interaction when it is received at the functional contract "*FC4*".

```
rule "UpdateRequestRouteContextInfromation"
    when
        $event : MessageRecievedEvent(operationName == "RequestRoute1")
    then
        MessageContextParamterUpdate I1ContextParamterUpdate=new MessageContextParamterUpdate();
        MessageWrapper mw = $event.getMessageWrapper();
        I1ContextParamterUpdate.updateMessage(mw, "CongestedRoads" );
        I1ContextParamterUpdate.updateMessage(mw, "BlockedRoads" );
end
```

Second, 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 or its functional contracts. 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 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.

Third, to enable the execution of the *adaptation rules and meta-rules* described above, we transform them to Drools 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. The result of transforming the "*AdaptatioRule1"* described above is shown blow. In transforming the rules, we used the rule "*When*" part to specify the rule *event* (e.g. the user need of the route planning service is changed). In addition, the rule "*Then*" part is used for capturing both the rule *condition* and *action*. To evaluate the rule condition part, we created a class called "*ConditionEvaluator*". This class has a method called "*evaluate*" that takes a condition as an input, and then it replace the condition variables with current context values and returns true or false based on the condition evaluation. When the condition is evaluated to true (i.e. the user wants the route planning service for example), a set of adaptation actions are added to the adaptation script (e.g. *actions.addAction("AddRole_RoutePlanner"*). By the same manner, the adaptation meta-rule "*AdaptatioMetaRule1*" can be generated.

```
rule "AdaptationRule1"
    when
        ContextEvent(  name == "RoutePlannerSelected_Value_Chaged" )
    then
        if(conditionEvaluator.evaluate(" ( RoutePlannerSelected == True ) ")){
            actions.addAction("AddRole_RoutePlanner");actions.addAction("AddContract_FC3");
            actions.addAction("AddContract_FC4");actions.addAction("AddContract_FC5");
            actions.addAction("AddRole_TrafficInformation");actions.addAction("AddContract_CC6");
            actions.addAction("AddContract_CC8");actions.addAction("Bind_RoutePlanner_RoutePlanner1");
            actions.addAction("Bind_TrafficInformation_RoadSideUnit");
        }
end
```

The adaptation rules are used to generate an adaptation script. This script is then used by the organizer player to adapt the running system by invoking the organizer role standard adaptation methods that are corresponding to the required actions. These standard methods require some details to execute. For example, to add a role there is a need for the role *name*,

[3] http://www.jboss.org/drools

identifier and *description*. Here comes the role the maintained runtime representation of the system models where they have these required details. As such, the organizer player parses these representations to get the required details and generate the executable actions. For example, adding the route planner role in the above script is transformed to *organizer.addNewRole("FR2", "RoutePlanner", "Role represents route planning service")*. The *organizer* variable is a reference to the running system organizer role. A similar mechanism is used for changing the adaptation rules, where we have a reference of the loaded adaptation rules (i.e. instance of KnowledgeBase[4] class) and the methods *addKnowledgePackages* and *removeRule* are used to add and remove adaptation rules respectively.

The above transformation process is automated in our tool. When the software engineer completes the system modelling, he can press a button that generates the files required by the ROAD framework to deploy an instance of the system. This instance contains the system *roles*, their *contracts* and the generated *organizer player*. To have a fully running system, we have developed a set of functional players and context providers. For example, we used Google maps[5] services to develop he *route planners, attractions finder, and restaurant locator* players. We also have developed a GUI to enable the user interactions with the provided services as shown in Figure 5.

In Figure 5-A, the application only includes the attraction finder service which is provided 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 application. As such, the application is adapted to include such service by performing the adaptations described above. After the internal adaptations are performed, the application GUI is changed also by including the route planning service. When this service becomes available, it acquires tourist location, his attractions list, and traffic information to suggest a suitable route (see Figure 5-B).

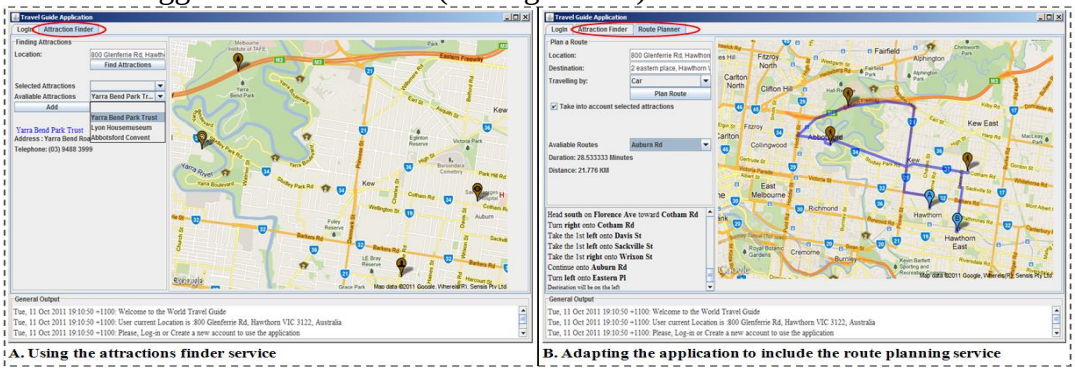

Fig.5. The context-aware adaptive travel guide application in action

4.3 Performance Evaluation[6]

The overhead in our approach is the extra time needed to adapt the system while it is in operation. This can be calculated by the time required to *monitor the context*, *decide the required adaptation actions*, and *act these actions*.

Monitor the context. In our approach, there is a need to keep track of some context variables that cause system adaptation. When any of the variables that the adaptation rules are interest in changes, this change is notified to the organizer player to decide the needed

[4] http://docs.jboss.org/jbpm/v5.1/javadocs/org/drools/KnowledgeBase.html
[5] http://code.google.com/apis/maps/documentation/webservices/
[6] A PC with Intel Core 2 Due 3 GHZ CPU and 3 GB RAM is used as the test-bed and Drools-5.1is used as the rule engine.

adaptations. The time required to notify the system organizer with a context variable change equals to *14.59 milliseconds* in average.

Decide required adaptation actions. When the context is changed, the adaptation rules need to be evaluated to decide the required adaptation actions. The overhead in the decision making process is laid in rules loading time at the beginning and their execution time in response to context changes. To measure that overhead, we used sets of rules with sizes 10, 20, 30, 40, and 50. Figure 6-A shows that the rules loading time varies from *1.81 to 2.05 seconds* based on the adaptation rules size. This is not much overhead where it is usually performed once at the system start-up. In addition, the time to execute the rules is between *14.8 to 29.9 milliseconds* (see Figure 6-B) which cannot be considered as an overhead also.

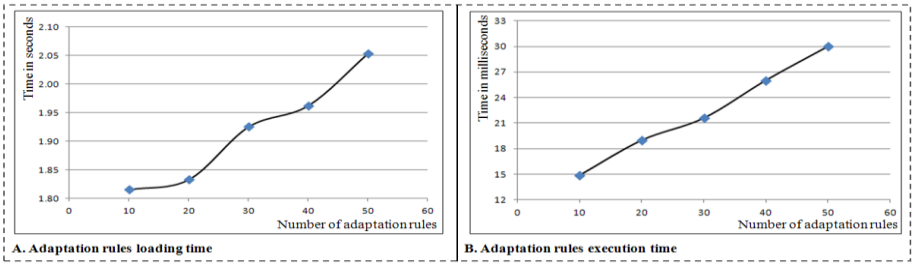

Fig.6. The adaptation rules loading and execution times

Apply the adaptation actions. We have different adaptation actions that can be performed to adapt the system in response to context changes. Table 1 summarises the average time needed to apply some of the required adaptation actions in milliseconds. In Table 1, we only show the actions for adding some elements to the system, where they are of interest from the user point of view (i.e. they are added where the user wants them). The adaptation actions for removing parts of the system have a small overhead and are performed while it is running. As such, they do not affect the user interactions with the system. Due to space limitation, we do not present them here.

Table 1. The time required to apply the required adaptation actions in milliseconds

Adaptation action	Required time	Adaptation action	Required time
Add Functional/Context Role	198.953	Change Role-Player Binding	0.0348
Add Functional Contract	0.204	Add Adaptation Rule	0.0096
Add Contextual Contract	1.565		

A delay that the tourist can experience in using the travel guide system is happened when he wants to include the route planning service. To include this service, it takes around *422 milliseconds* which cannot be considered as a delay to the tourist.

5. RELATED WORK AND DISCUSSIONS

A number of approaches have been proposed to support the process of developing context-aware adaptive systems. Some of them propose a way to *model* and *realize* such systems [4, 8, 14, 25-27, 29] while the others provide a framework or a middleware to help the software engineer in *realizing* the context *acquisition* and *interpretation* [5, 13, 28] or the system *runtime management* [1-3, 7, 10]. In this section, we analyse existing and our approaches in relation to the requirements we have identified in section two.

Compose a context-aware system: Existing approaches to developing context-aware adaptive systems can be classified into two categories. On the one hand, a set of approaches consider the system functionality as a single service and the context information is used to

adapt this service operational *parameters* [27, 29]. On the other hand, a set of approaches has been proposed to compose a system from a set of components that are changeable at runtime to cope with context changes [4, 8, 14, 25-26]. However, they capture the *relationships* between the system and its context implicitly during the system development except few who only consider these relationships explicitly at design time [4, 27, 29]. As such, the system implementation complexity is increased and the system-context relationships changes become difficult and error prone. In addition, in composing system's functionality, existing approaches connect the composed components directly (i.e. linking the components required and provided interfaces directly) [4, 8, 14, 25-26]. As such, in large scale systems, the composed elements complex interactions and their mutual obligations become difficult to capture and manage. In our approach, we *explicitly represent the relationships* between the system and its context and between the system functional elements themselves. As such, we can clearly *compose and realize* a context-aware adaptive information system. Similar to our approach Sheng et al. [27] propose approach who only consider these relationships explicitly only at design time and they also consider the system as a single service

Runtime adaptation of the context model: At runtime the context model may change by including a context attribute or remove an existing one as shown in our motivating scenario. However, most of existing approaches have only a design time context model and usually it disappears during the system implementation where it is intertwined with system's functionality and/or management. Few approaches keep the context model explicit at runtime [26, 28], so that they can switch between different context providers or include new providers while system is in operation. However, they do not have the ability to change the context model itself by adding, removing, or modify the system required context attributes. Our approach has an adaptable *runtime representation* of the context model and its management enables its *runtime changes*.

Adaptation logic runtime changes: Existing approaches use *goal-policy* [14, 29], *utility-policy* [8, 26], or *action- policy* [1-2, 13] to capture the system *adaptation logic*. Most of these approaches do not support the runtime changes to the system adaptation logic except Andrade et al. [2] who enable the system developer to change this logic manually. However, the adaptation logic of the travel guide system need to be changed automatically without the developer involvement as the system may include or exclude parts of its adaptation rules while it is running. To do so, our approach maintains a *runtime representation of the adaptation rules* and we have a set of *adaptation meta-rules* that decides the required changes to the adaptation rules in response to context changes. In this paper, we adopted the *action-policy* approach in capturing the adaptation logic because of its *expressiveness* and *availability* of tools that supports *runtime changes* of the rules.

6. CONCLUSION

In this paper, we have proposed a model-driven approach to *developing* context-aware adaptive information systems. We have considered the system model, the context model, and their relationships explicitly from *modelling* to *realization* and to runtime *execution* and *adaptation*. In addition, we have developed a prototype tool for modelling the system and generating its implementations from their models. Furthermore, we have demonstrated our approach through the *development* of the context-aware adaptive travel guide system. We also *measured* the overhead of adding the runtime adaptability feature to a software system using our approach.

Compared to existing approaches, our approach has the following key contributions. Firstly, we explicitly *separate but relate* the system model and the context model, so that their relationships and changes can be clearly captured and managed. Secondly, the *relationships*

between the system functional elements are represented explicitly, so that the functional elements interactions, mutual obligations, and changes can be clearly captured and managed. Thirdly, our approach supports the runtime adaptation of the *system*, the *context model*, and their *adaptation logic* in response to *context changes*. Finally, our tool supports the system modelling and the generation of its implementations from their models.

As future work, our approach can be enhanced in several directions. First, software systems are usually deployed in environments which are not totally anticipated at the system design time. While we have a runtime representation of the system aspects (including rule-based management) to be able to cope with unanticipated changes, runtime system management strategies and decision-making techniques are required to fully realize this capability. Second, we have applied our approach to the tourist travel guide case study, and the results were promising. We will perform more validations to assess the applicability and practicality of our approach.

REFERENCES

[1] Adamczyk, J., Chojnacki, R., Jarząb, M., & Zieliński, K. (2008). Rule Engine Based Lightweight Framework for Adaptive and Autonomic Computing *International Conference on Computational Science* (Vol. 5101, pp. 355-364).

[2] Andrade, S. S., & de Araujo Macedo, R. J. (2009, 18-19 May 2009). *A non-intrusive component-based approach for deploying unanticipated self-management behaviour*. Paper presented at the Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS '09. ICSE Workshop on.

[3] Asadollahi, R., Salehie, M., & Tahvildari, L. (2009, 18-19 May 2009). *StarMX: A framework for developing self-managing Java-based systems.* Paper presented at the Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS '09. ICSE Workshop on.

[4] Ayed, D., Delanote, D., & Berbers, Y. (2007). *MDD approach for the development of context-aware applications*. Paper presented at the Proceedings of the 6th international and interdisciplinary conference on Modeling and using context, Roskilde, Denmark.

[5] Capra, L., Emmerich, W., & Mascolo, C. (2003). CARISMA: context-aware reflective middleware system for mobile applications. *Software Engineering, IEEE Transactions on*, *29*(10), 929-945.

[6] David, P.-C., & Ledoux, T. (2006). An Aspect-Oriented Approach for Developing Self-Adaptive Fractal Components *Software Composition (SC'06)* (Vol. LNCS 4089, pp. 82-97).

[7] Elkhodary, A., Esfahani, N., & Malek, S. (2010). *FUSION: a framework for engineering self-tuning self-adaptive software systems*. Paper presented at the Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering, Santa Fe, New Mexico, USA.

[8] Floch, J., Hallsteinsen, S., Stav, E., Eliassen, F., Lund, K., & Gjorven, E. (2006). Using Architecture Models for Runtime Adaptability. *IEEE Softw.*, *23*(2), 62-70. doi: http://dx.doi.org/10.1109/MS.2006.61

[9] France, R., & Rumpe, B. (2007). *Model-driven Development of Complex Software: A Research Roadmap*. Paper presented at the 2007 Future of Software Engineering.

[10] Garlan, D., Cheng, S. W., Huang, A. C., Schmerl, B., & Steenkiste, P. (2004). Rainbow: architecture-based self-adaptation with reusable infrastructure. *Computer*, *37*(10), 46-54.

[11] Giachetti, G., Marín, B., & Pastor, O. (2009). Using UML as a Domain-Specific Modeling Language: A Proposal for Automatic Generation of UML Profiles *Advanced Information Systems Engineering* (Vol. LNCS 5565, pp. 110-124).

[12] Governatori, G. (2005). Representing Business Contracts in RuleML. *International Journal of Cooperative Information Systems, 14*(2), 181-216.

[13] Gu, T., Pung, H. K., & Zhang, D. Q. (2005). A service-oriented middleware for building context-aware services. *J. Netw. Comput. Appl., 28*(1), 1-18. doi: http://dx.doi.org/10.1016/j.jnca.2004.06.002

[14] Heaven, W., Sykes, D., Magee, J., & Kramer, J. (2009). A Case Study in Goal-Driven Architectural Adaptation *Software Engineering for Self-Adaptive Systems* (pp. 109-127): Springer-Verlag.

[15] Jin, Y., & Han, J. (2005, 15-17 Dec. 2005). *Consistency and interoperability checking for component interaction rules.* Paper presented at the Software Engineering Conference, 2005. APSEC '05. 12th Asia-Pacific.

[16] Kapuruge, M., Colman, A., & King, J. (2011, Aug. 29 2011-Sept. 2 2011). *ROAD4WS -- Extending Apache Axis2 for Adaptive Service Compositions.* Paper presented at the Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International.

[17] Kelly, S., & Tolvanen, J. P. (2008). *Domain-specific modeling: enabling full code generation*: Wiley-IEEE Computer Society Pr.

[18] Kramer, J., & Magee, J. (2007). Self-managed systems: an architectural challenge. *Future of Software Engineering, 2007. FOSE'07*, 259-268.

[19] Liaskos, S., Litoiu, M., Jungblut, M., & Mylopoulos, J. (2011). Goal-Based Behavioral Customization of Information Systems. In H. Mouratidis & C. Rolland (Eds.), *Advanced Information Systems Engineering* (Vol. 6741, pp. 77-92): Springer Berlin / Heidelberg.

[20] Ludwig, H., Keller, A., Dan, A., King, R. P., & Franck, R. (2003). Web service level agreement (WSLA) language specification. *IBM Corporation*, 815-824.

[21] Maturana, H. R., & Varela, F. J. (1987). The tree of knowledge the biological roots of human understanding. *1st ed edn. Boston: New Science Library. Distributed in the United State by Random House*.

[22] McCarthy, D., & Dayal, U. (1989). The architecture of an active database management system. *ACM SIGMOD Record, 18*(2), 215-224.

[23] Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys (CSUR), 37*(4), 316-344.

[24] Mintzberg, H. (1994). Rounding out the manager's job. *Sloan Management Review, 36*, 11-26.

[25] Morin, B., Barais, O., Nain, G., & Jezequel, J.-M. (2009). *Taming Dynamically Adaptive Systems using models and aspects*. Paper presented at the Proceedings of the 31st International Conference on Software Engineering.

[26] Rouvoy, R., Barone, P., Ding, Y., Eliassen, F., Hallsteinsen, S., Lorenzo, J., . . . Scholz, U. (2009). MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments. In B. Cheng, R. de Lemos, H. Giese, P. Inverardi & J. Magee (Eds.), *Software Engineering for Self-Adaptive Systems* (Vol. 5525, pp. 164-182): Springer Berlin / Heidelberg.

[27] Sheng, Q. Z., Jian Yu, Segev, A., & Liao, K. (2010). Techniques on developing context-aware web services. *International Journal of Web Information Systems, 6*(3).

[28] Taconet, C., Kazi-Aoul, Z., Zaier, M., & Conan, D. (2009). *CA3M: A Runtime Model and a Middleware for Dynamic Context Management*. Paper presented at the Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, Vilamoura, Portugal.

[29] Zhang, J., & Cheng, B. H. C. (2006). *Model-based development of dynamically adaptive software*. Paper presented at the Proceedings of the 28th international conference on Software engineering, Shanghai, China.

