Context-Aware Reconfiguration of Software Systems

Rima BOUDIDA, Faiza BELALA, Chafia BOUANAKA
Computer Science Department
Mentouri University
Constantine, Algeria

Rima.boudida@hotmail.fr, belalafaiza@hotmail.com, c.bouanaka@umc.edu.dz

Abstract
The specification and the design of software systems that dynamically adapt their behavior at runtime in response to changes in their requirements, user preferences, operational environments, and underlying infrastructure, poses a great challenge. In this paper, we propose a unique model that supports formal definition of the required and provided functionalities of such systems, their context information and management actions. With the objective to gain a formal specification of the relationships between these three aspects, we integrate our developed model in rewriting logic framework and execute it under Maude environment while exploiting its strategy language. A strategy expression in Maude allows the control of rewriting process of a given term.

Keywords: Adaptive Software Systems, Context-aware Reconfiguration, Maude.

1. Introduction
Nowadays, self-reconfiguration of software systems constitutes an essential property in the design and implementation of robust and evolutionary systems. There is an increasing need for software systems that dynamically adapt their behavior at runtime in response to changes in their requirements, user preferences, operational environments, and underlying infrastructure. A challenge is how to specify and verify such systems in order to realize them.

Various researches in the area of systems that adapt themselves in response to context and/or requirements changes have been conducted by researchers. We highlight two classes of works with two different emphases: self-reconfigurable systems [1] and context-aware systems [2],[3],[4]. In context-aware systems, researchers are more concerned with how to model contextual information of a system; they do not pay any attention to show how this system adapts itself in response to (unanticipated) changes in the context information or how it manages requirements changes while it is in operation. Researches in self-reconfigurable systems are more worried about how to adapt the system in response to context and/or requirements changes by separating system functionalities from its management. In this case, no model has been given for context.

In practice, it is necessary for a software system to change itself as necessary to continue achieving and/or preserving its new and existing goals that evolve at runtime. The design of a context aware adaptive system needs to consider both aspects in a consistent manner.

In this paper we propose a unique model that explicitly supports formal definition of the required and provided functionalities, context information and management actions, all involved in the design of component based software applications. With the objective to formally specify the relationships between these aspects, we integrate our proposed model in rewriting logic framework and execute it under Maude environment [5]. Maude (SRI laboratory, United States) is a declarative language where several concurrent applications have been considered. Furthermore, it also offers various simulations, search, and model checking techniques for discovering erroneous behaviors in system specifications.

Our main contribution consists in exploiting Maude strategy [6] language to define how context changes affect the structure and the behavior of a system. This language allows the definition of strategy expressions that control the way a term is rewritten.

The remainder of this paper is organized as follows. We begin by reviewing some existing works related to ours. Then, we introduce in section 3 the proposed model to deal with adaptive and context aware component based systems. Maude object oriented modules are also built to implement our developed model. Based on a case study illustrating the proposed formalization approach, we show the relevance of our solution to model and manage context information for various reconfiguration actions in section 4. Finally, we conclude the paper with constructive remarks and eventual perspectives.
2. Related work

Various models exist in the literature for the specification of component-based systems adaptation; we will study three most significant of them that treat context-aware dynamic adaptation.

The first model [7] uses architectural graphs for the description of component-based systems: nodes represent functional components and links correspond to communication ports.

System graph is intended to evolve via graph rewriting rules; specifying structural changes caused by system dynamic reconfiguration. These rules allow graph transformations, sub-graphs addition/deletion, and are composed of two essential parts: structural and functional.

- The structural part is defined using an integrated Δ-notation whose body specifies structural changes to be operated on the graph: retraction (deletion), insertion and restriction if some guards and required context are verified.

- The functional part expresses preconditions of transformation rule application with respect to system properties that are intended to monitor changes. It also specifies functional actions to be performed, if the transformation rule is applied.

The semantics associated to each graph transformation rule is a diagram of Z-operations.

The second model [8] is based on hypergraphs to describe component-based systems where components and connectors are represented via hyper-edges and communication ports through nodes.

To formally describe the process of dynamically reconfiguring component-based software systems, hypergraphs production rules are used. Each production rule has the following form G(H, p, H0), meaning that the hypergraph H0 is transformed to H by applying p which is a set of addition/deletion operations denoted by symbols ('U' and '−') respectively.

The third model [9] combines box/line notation and Petri nets to specify context-aware dynamic software systems. System architecture is described using a boxes/lines notation with boxes corresponding to components and lines to communication ports that can be of three types: functional, adaptation, or contextual.

System behavior is specified using Petri nets.

The model is structured into three layers. The functional layer describes system architecture via functional components and their interconnection links. The contextual model specifies contextual information pertinent to system adaptation. The making decision layer is responsible of triggering adaption operations on functional components according to contextual information evolutions.

The three models presented above treat dynamic reconfiguration of component-based software systems using different approaches and formalisms. While both of the 1st and 2nd models are based on a mathematical basis, respectively Graphs and Hypergraphs as they are relevant for structural representation, the third model is based on a graphical tool named CAST to design the different components that are supported by the system considered without any formalization.

Models 1 and 2 provide a formal and graphical framework to specify structural aspects of components based systems. However, both graphs and hypergraphs lack notations to specify quantitative or functional properties of software systems. To overcome this lack, model 1 interleaves the Z language to graphs where logical expressions are used to specify both functional and quantitative properties of component-based systems.

Concerning dynamic reconfiguration, the first two models adopt graph transformation rules to dynamically rewrite graphs/hypergraphs. In addition, model 1 offers necessary notations to specify allows constraints that might be verified during the reconfiguration process in order to ensure system consistency during its evolution via Z specifications. However, model 3 has no formal model to specify dynamic reconfiguration of software systems. It just comply the approach proposed by the author.

We conclude this discussion, see table 1 below, by highlighting the following remarks: Although, graphical aspect of graphs/hypergraphs in models 1 and 2 facilitate the specification and readability of system architecture, these models engender an exponential complexity compared to the number of components/connectors. The absence of reconfiguration triggers in models 1 and model 2 complicates the expression of context changes that provoke dynamic reconfiguration of graph/hypergraph fragments. In contrast, model 3 supports the description of contextual information; it lacks formal approach to specify component-based systems.

Our objective in this work is to propose a formal model for the specification of context-aware adaptive systems based on Maude language.
3. Dynamic adaptation guided by the context

Our objective in this paper is to present a specification model for context aware adaptive systems. The model is composed of three levels (see Fig. 1):

The base level contains functional components that are responsible of realizing system core functionalities. System management level is a controller that; according to contextual information changes decides what adaptation rule to execute.

The third level is composed of a set of ports (environment variables) allowing the system to capture contextual information that functional components need to perform some actions or system management layer expects to trigger an adaptation action. Context level brings context information from the external environment.

There exist two kinds of connection (inter & intra); Links of type 1 (see Fig. 1) represent “Intra” connections, they serve to attach components of the same level. This kind of connections can be found only in the first level where functional components need to communicate in order to exchange their provided services.

Links of types 2, 3, 4 represent “Inter” connections, they attach components of different levels;


Links of kind 2 serve to attach functional components to context components in order to provide contextual information, necessary for performing some actions, to the functional level.

Links of kind 3 are used to connect system management components to context components. They provide contextual information, necessary for triggering adaptation decisions, to system management components.

Links of kind 4 attach functional components to system management components. According to adaptation decisions adopted in system management layer, some adaptation actions (addition, deletion, replacement of functional components) are to be applied on functional components and are propagated to the functional level via this kind of connections.

Formal specification of context aware adaptive systems structure and components is realized by associating a HyperGraph to each level. The whole system is also considered as a Hypergraph obtained by composing basic Hypergraphs with respect to Inter-level connections.

3.1. Functional components:

A functional component, see Fig. 2, has three types of ports:

Table 1. Comparison of component based adaptive systems.

<table>
<thead>
<tr>
<th>Model</th>
<th>Software architecture specification</th>
<th>Functional specification</th>
<th>Reconfiguration</th>
<th>Verification</th>
<th>Context aware adaptation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 1</td>
<td>Graphs</td>
<td>Z specification</td>
<td>Graph rewriting rules + Z specification</td>
<td>Tool supporting Z notation</td>
<td>NO</td>
</tr>
<tr>
<td>Model 2</td>
<td>Hypergraphs</td>
<td>No</td>
<td>Production rules</td>
<td>Model Checker</td>
<td>NO</td>
</tr>
<tr>
<td>Model 3</td>
<td>CAST tool</td>
<td>No</td>
<td>CAST tool</td>
<td>Petri Net + Temporal Logic</td>
<td>YES</td>
</tr>
</tbody>
</table>

Fig1. A layered model for context aware adaptive Software systems
Functional port: represents Intra links and serves to attach a requested service of some functional component to a provided service of some other one.

Context port: forwards contextual information delivered by a context component to the functional one.

Adaptation port: adaptation actions that the functional component can support are provided to systemManagement components in order to take the adequate adaptation action.

The two latter ports represent ‘inter’ level connections, i.e., of kinds 2 and 4 respectively and presented above. Each Hyperedge (functional component) could be a whole HyperGraph since it may be a composite or hierarchical component.

3.2. System Management components:

A systemManagement component, see Fig. 3 below, has two kinds of ports "contextual information" & "adaptation", it carries contextual information from a context component, performs some adaptation decisions and then transmits adaptation actions to functional components. These links are Inter level connections since they connect SystemManagement components to components from other levels. Each component (HyperEdge) could be an entire HyperGraph since it could be a composite component.

3.3. Context components:

A context component, see Fig. 4 below, has a unique kind of ports "contextual information" as required and provided ports. It requires such information from realizations of context, i.e., execution environment, and forwards it to functional or SystemManagement components. A Context component has only inter-level links.
3.4. Maude specification of HyperGraphs:

In order to specify HyperGraphs, HyperEdges and the different ports and links between them, we employ rewriting logic and its implementation language Maude. We use Maude object modules to express each concept. We begin with the specification of a Hypergraph, defined by a set of Hyperedges and nodes. We define a class Hyperedge which is a class characterized by a set of input port and a set of output ports.

After that we specify how a Hypergraph should be.

As we have explained previously in section 3, each type of components belonging to our model is a hypergraph. For example, a functional component extends the Hypergraph class by equations defining nodes connections.

A Functional component contains four classes. While the first one defines the functional component itself, the other ones represent the three different types of ports, corresponding to nodes in the Hypergraph, connecting the functional component to the others components (context, management and other functional components).

The second module specifies a context component which has two classes, one defining the component itself and the second one of type node defines the different ports(nodes) connecting the context component to the other ones (functional & management components).

The third module specifies a SystemManagement component which is declared in a similar way as context components.

The whole system is obtained by instantiating the different types of component modules and establishing inter level links between them. A link in our specification model is a rewrite rule with a left hand side specifying an output port of a component and the right hand side specifying the input port of another component.
4. Case study

To illustrate our proposed model, we choose the PMS system (Patient Monitoring System), composed of patients, controllers and nurses. A controller must be tied to each patient bed to take the necessary measures. Thanks to this controller, each nurse can remotely check the status of her patients by asking information about the blood tension, temperature, etc…

The PMS is a scalable system with the following properties:
- The maximum number of services is 3.
- A service contains a maximum of five nurses.
- A service contains at maximum 15 patients.
- A patient should always be affected to a single service.

Figure 5 below presents the component based system associated to the PMS system:

To ease the readability of the figure, we have not considered all system states. We have taken an instance with three functional components, two contextual components and two SystemManagement components.

An instance of the PMS system contains three types of functional components: patient, event_service and nurse, the contextual model contains two components add_nurse and disconnect_patient, the SystemManagement component has two components applyNurse and changePatientService. The second contextual information considered is a patient need to change its current service, the SystemManagement component reacts by disconnecting the patient from the actual service and connecting him the destination service.

Maude specification of the PMS system is as follows:

Nurse Component

(omod NURSE including STRING .
inc FunctionalComponent .
inc ContextComponent .
inc ManagementComponent .
inc link .
subclass nurseComponent patient < Component .
msg _request_from_ : Oid String Oid -> Msg .
var nurse1 : Oid .
var serv1 : Oid .
var O : Oid .
var I : Oid .
var S : String .
rl [request1] : (nu1 request S from pt)
< nurse1 : nurseComponent | outPort : O > <
O : Node | name : N , ser : S >
=> < pt : patientComponent | intPort : I > < I :
Node | name : N > .
endom)
Since the Maude engine is based on a non-nondeterministic semantics, we are not able to ensure that a contextual information reception by a context component enforces an immediate reaction of the SystemManagement component if an adaptation is necessary. For example, the arrival of a nurse detected by the context component via its `Transfer` rule execution might be followed by the addition of a nurse by the SystemManagement component. Therefore, the system designer is constrained to explicitly enforce such scenario via the definition of a Maude strategy, specifying rules execution order. As mentioned in Section 1, our main contribution is the exploitation of the Maude strategy language in order to formally define relationships between context information, management actions, and system functionalities. We use Maude strategies to specify the way Maude will execute the different rules according to the different changes in the context.

The module below defines two strategies for adding a new nurse and changing patient service respectively.

The strategy `applyTransfer` controls nurse addition by ensuring the sequential execution of rules.
Transfer of context component and Apply of SystemManagement component. Whenever add_nurse component receives manageapplyTonurse message, rule Transfer of context component is executed (transferring the information to SystemManagement component); sequentially rule Apply then applied to add a new instance of nurse component.

The second strategy changeServiceOfExistingPatient ensures executing a patient disconnection from the actual service and his affection to a new nurse in the destination service.

```
(stratdef mystrat is
  inc Nurse.
  inc Patient.
  inc Link.

  strat applyTransfer = if < ctxt : Context |
                   outPort : O > < O : Node | name : N,
                   ctxt_information : info > then transfer ;
                   Apply else idle fi.

  strat changeServiceOfExistingPatient = if <
                  patient : patientComponent | service_name : S1,
                  outPort : P > then ChangePatientService
                  else idle fi.
endsd)
```

5. Conclusion

Various researches in the area of self adaptive systems in response to context and/or requirements changes have been conducted. In this paper, we have considered three significant models that treat self adaptation. The major remark is that these works have not successfully answer when and how to perform a self adaptation. don’t treat self reconfiguration in a correct way. While, some models consider only how to realize dynamic adaptation by given more importance to a mathematical tool to specify adaptive system, and completely ignore the importance of contextual information for system self reconfiguration, other models consider context (when) as an important aspect but do not define it in a formal way.

In our work, we tried to define a formal model for context aware adaptive system based on Maude language and its stategy language. The model explicitly supports formal definition of the required and provided functionalities, context information and management actions, all involved in the design of software component based applications. Our major contribution consists of formally specifying the relationships between these three aspects using strategies.

As future work, we plan to improve the model by introducing other strategies considering other self adaptation cases. Once the specification model finalized, it is necessary to analyze and verify it.

References


