A component-based approach for Context-Aware Systems specification

Brahim Djoudi, Chafia Bouanaka, Nadia Zeghib
LIRE Laboratory
University of Constantine 2
Constantine, Algeria.
DjoudiBrahim@hotmail.fr, c.bouanaka@umc.edu.dz, n_zeghib@hotmail.com

ABSTRACT
Software systems adaptation is almost an inevitable process, due to changes in customer requirements, needs for faster development of new, or maintenance of existing services, etc. System adaptability is its ability to change its structure and/or behavior in response to changes in the environment. This paper proposes a formal model for specifying context-aware self-adaptive applications. The approach is based on Maude language and takes advantages of its reflection feature to specify in the same formal framework both structural and behavioral aspects of software systems. A component-based approach is adopted where system components are specified in an entirely independent manner from context ones and only context changes impact on system functionalities and structure is specified. Thanks to this model, we promote reusability and facilitate extensibility of both system elements and context ones. Moreover, formal analysis may be easily performed using Maude model checker.

Categories and Subject Descriptors
D.2 [**Software Engineering**]

General Terms
Algorithms, Design, Reliability, Languages, Theory

Keywords
Self-adaptation, Context-aware systems, Formal methods, Maude, Meta-programming.

1. INTRODUCTION
Nowadays, the main feature of ubiquitous applications is their adaptability at different phases of their execution life cycle. System adaptation is indispensable in such applications since they are exploited by different users with distinct profiles and executed in various platforms. Thus, software system needs to change itself as necessary to continue achieving and/or ensuring existing and new goals that evolve at runtime. Besides, it needs to adapt to runtime environment variations. Thus, a ubiquitous application that might be able to take into account the execution context to adapt to its variations is called context-aware.

The basic idea of context-aware adaptive systems is that they anticipate –based on context– what system functionalities the user will need and execute. Technically, this context information is sometimes referred to as triggers that launch an adaptation.

Several models and architectures have been proposed to develop context-aware software systems. They are all concerned with how to model, process, and manage the context information. Generally, models of context-aware adaptive systems are proposed without paying attention to reusability aspects of both system and context entities. Additionally, the proposed models are not able to represent context information and its relationships with system entities without affecting the system complexity and consistency.

We aim in this paper to propose a formal model for context-aware adaptive systems that establishes a clear separation of concerns between system components and context entities. The goal is gained by the definition of a layered model where functional system and context layers are designed in an entirely independent manner and only relationships between them are dynamically established via the generation of strategies by the management layer.

The model is composed of three layers, where the top level element is the context model that serves to specify contextual entities relevant to system operation and/or adaptation. The second level is the management model which is responsible of transforming the impact of context changes on functional system structure and behavior by applying different reconfiguration strategies; that a parameterized by context values changes and the associated actions on system structure and behavior. Adaptation actions execution reflects context changes side effects on system functionalities in terms of variations on system structure and/or behavior giving the management layers a high level of genericity and reusability.

Maude is adopted as a semantic framework for the proposed model. We exploit Maude reflection and meta-programming capabilities to enrich it with context-awareness concepts.

The remainder of the paper is organized as follows: Section 2 discuses some models for context-aware systems specification that relevant to our work. Section 3 recalls basic concepts on meta-programming and strategies in Maude. Section 4 is dedicated to the presentation of our model. Finally, Section 5 concludes the paper with ongoing work.

2. RELATED WORK

Numerous researches have been defined and various context-aware middleware have been proposed in order to simplify the development of adaptive context-aware applications.

Aura [12] is an appropriate architecture for ubiquitous computing. This system is based on the idea of distinctive atmosphere (Aura) being generated by someone or something to act as a proxy for the user it represents. When changes in user environment are produced, Aura provides a support for user tasks by adapting to local resources. SOCAM [13] is a middleware providing a support for most of the tasks concerned with the context such as getting their context from various sources: the context of interpretation, and sharing of context. However, the reuse of a context and the relationship between the system and the environment are not taken into account.

Andrade et al [1] proposed a platform based on components to provide a self-management of deployed applications, using a non-intrusive approach that supports unanticipated architectural changes. The platform includes components for specifying adaptation strategies, environmental monitoring, implementation of the devices, and redeployment of architectural changes. Self-management is carried out by two separate but related configurations of components: the application specific system and autonomous observation system. Nevertheless, this approach does not consider the system behavioral changes.

MDD [2] (Model Driven Development) is a semi-formal modeling approach proposed for the development of context-aware applications. An implementation can be generated from the model in a straightforward manner. UML diagrams were extended with new stereotypes to undertake elements of context sensitivity. The MDD framework provides a clear solution, but does not make available reusable infrastructure (a specific model for each system). In addition, the semi-formal representation (UML) decreases the accuracy and consistency of the model.

Safran [8] is an extension of the Fractal component model aimed as a support to self-adaptive components development. Safran allows components to automatically reconfigure their structure according to context changes. However, this model is very tightly related to a specific model that is fractal. Moreover, only reconfiguration of existing components is allowed.

Rainbow [11] Framework provides mechanisms for monitoring the functional system and its context and allows selecting necessary strategies for adaptation. It is based on Acme to model the system and its context. A system context is represented as a set of resources that support the adaptation process. Unfortunately, the model has a poor representation of context information and its relationships with system entities.

A variety of research work has been realized in the literature on context-aware systems verification. Based on Role-Oriented Adaptive Design (ROAD) framework, authors in [15] propose ROAD4Context approach to model context-aware interactions, abstract processes and invariant properties. System functional behavior is modeled using UML-based process models. Such models are then transformed into Petri nets-based formal models and system invariants into linear temporal logic. In [9], authors propose a mapping process of static and dynamic aspects, described by UML class diagram and UML state and communication diagrams respectively, of object-oriented systems to Maude concurrent object-oriented systems. The so developed model properties are verified using Maude LTL model checker.

In [16], authors propose an approach for modeling context-aware services equipped with a toolkit for application development facilities. The proposed methodology adopts Colored Petri Nets with some rule extensions, and algorithms for timely context-aware service. In [7] authors propose a prototype language CDL (Context Description Language) for formalizing contexts and properties. CDL permits contexts and non-ambiguous properties to be formalized and checked using simulation and model checking techniques.

In this work, we propose a framework for context-aware adaptive software systems specification that supports system and context modeling and system adaptation. The proposed framework defines context model elements in an independent manner from system model. This separation of concerns reduces system modeling complexity and increases model reusability and maintainability. Our model for adaptive context-aware systems provides a clear representation of context information and its relationships with system entities.

3. MAUDE LANGUAGE

Maude [6] is a high-performance language and system supporting both equational and rewriting logic specification and programming for a wide range of systems and applications. It also offers various simulation, search, and model checking techniques to detect erroneous behaviors in system specifications.

A key characteristic of Maude is its reflective tower having core Maude as its basic layer on which new paradigms and concepts can be defined (like it has been done for full-Maude, RT-Maude, Mobile-Maude) using Maude syntax itself. Such key functionality of reflection has been efficiently implemented in the functional module META-LEVEL. This module includes the modules META-MODULE for defining meta-modules and META-TERM for specifying meta-terms. Maude also allows reasoning at the Meta-level by proposing a variety of reduction and rewriting functions to operate reductions and calculus on meta-terms as `metaReduce`, `metaApply`, `metaXapply`, `metaRewrite`, `metaFrewrite`, `metaMatch`, and `metaXmatch`. These functions are called descent functions, since they allow us to descend levels in the reflective tower.

The META-LEVEL module introduces two polymorphic functions. The `upTerm` function takes a term t as its argument and returns the meta-representation of its canonical form. The `downTerm` function takes the meta-representation of a term t as its first argument and a term t_0 as its second argument, and returns the canonical form of t, if t is a term in the same kind as t0; otherwise, it returns the canonical form of t_0.

One most significant characteristic of Maude language is the possibility of controlling rewrite rules application, via the specification of strategies. Instead of applying a system of rules as a normalization procedure, a Maude strategy defines how to apply the rules. Generally, strategies are defined in extensions of the META-LEVEL module.

A strategy [14] is described as an operation that, when applied to a given term, produces a set of terms as a result, given that the process is nondeterministic in general. The basic strategies consist of applying a rule (identified by the corresponding rule label) to a given term; the rule may contain variables to be instantiated. For conditional rules, rewrite conditions can also be controlled by

means of strategies. Basic strategies can be combined by means of several combinators, including: regular expression constructions (concatenation, union, and iteration), if-then-else, combinators to control the way sub-terms of a given term are rewritten, and recursion.

The metaApply, and metaXapply operations are used to force applying particular rules:

```
op metaXapply :
Module Term Qid Substitution Nat Bound Nat
      ~>  Result4Tuple? [special (...)] .
```

where the first argument is the term to be reduced, the second argument is the rule label to be applied, and the third argument is a substitution function of eventual variables.

`metaParse` is a syntax verification operation, it reflects the parse command in Maude; that is, it tries to parse the given list of tokens as a term of the given type in the module given as first argument.

```
op metaParse :
Module QidList Type? ~> ResultPair?
                        [special (...)] .
```

Maude language is used not only to define domain-specific languages or tools, but also to build an environment for the given language or tool [5]. In such applications, while the predefined module META-LEVEL allows the definition of new programming constructs or language grammar, the LOOP-MODE module can be used to handle the input/output and to maintain the persistent state of the language environment or tool. A Loop-Mode object is defined by an input stream, an output stream and a system state. It has the following structure:

```
<input, state, output>
```

System state is a user definable structure, allowing a great degree of flexibility, and its evolution depends on inputs that are handled by strategies definition.

4. A MODEL FOR CONTEXT-AWARE ADAPTIVE SYSTEMS SPECIFICATION

Several context models have been defined and various context-aware middleware have been proposed in order to simplify the development of adaptive context-aware applications. Unfortunately by using specific middleware, those applications are not portable. This reduces considerably the possibility of reusing such applications in other projects since their development is tightly dependent on the target platform. In this paper, our main technical contribution is the definition of a generic formal specification framework for dynamically reconfigurable context-aware adaptive systems.

We propose a layered model (see Fig 1) that supports systems and system and context modeling in an independent manner. This separation of concerns reduces system modeling complexity and increases model reusability and maintainability.

4.1 Contextual Data Sources

Contextual information is obtained from different sensors. Sensors can be classified [2] in three types:

- Physical sensors: represent hardware sensors capable of capturing physical data
- Virtual sensors: these sensors collect context data from software applications.
- Logical sensors: these sensors use several context sources, and combine physical and virtual sensors with additional information in order to derive higher llayer evel context. They can interpret and reason about context information.

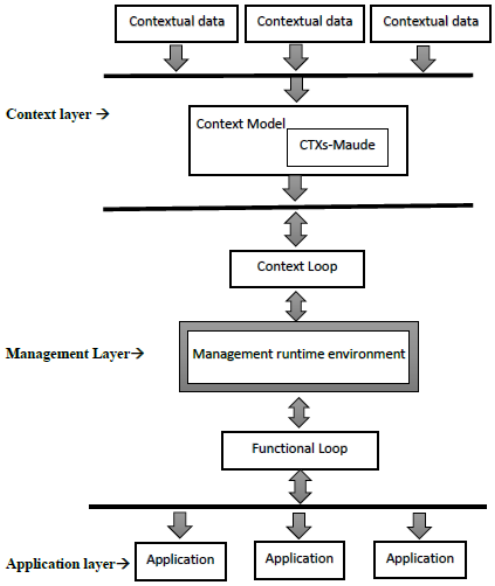

Figure 1. Context-aware systems model.

We are not interested on sensors function, only context information is relevant to our model.

4.2 Context Layer

The most interesting application domain of Maude language is that it allows modules with user-definable syntax. We exploit such characteristic to propose a new domain specific language, called CTXs-MAUDE (**C**on**T**e**X**t-aware systems using Maude) for a Maude-based language for contexts and adaptation actions specification and execution. The grammar of the language is defined as a layer on top of Maude by exploiting its reflection and meta-programming properties.

The Context layer contains all entities relevant to system operation and/or adaptation. The context model is defined as a set of context components to be declared in a context module in CTXs-Maude as follows:

```
CTXmod IdMod is /*Context module.*/
     /*Context, States...declaration.*/
endctxm
```

Each context component represents a concrete context; consisting of any information that can be used to characterize states of any entity considered as relevant to the interaction between users and the application and has an impact on system structure and behavior. We define a context component by:

- Context identity or context type such as the location zone of the user, and user preferences.

- Context description permits context definition and eventually determines the context sources that will be used to collect this information.
- Context States: various context states having an impact on the underlying application. Each state is defined by:
  - Context Value: each value acts as a trigger for system adaptation.
  - Actions: a set of actions to be performed whenever the context state is reached. System designer is responsible of specifying what, when and how the system reacts if the context value appears.

A context element is defined in CTXs-Maude as follows:

```
Context IdContext is
  CTXDescrip: /*Context description.*/
    CTXState: /*Context States*/
      State:
        CTXValue -> /*Context Value.*/.
        Actions:
           /*Action declaration.*/.
        endact
      endState
  ...
 endctxState
endctx.
```

Some adaptation operations require the presence of two or more different contexts states. This is declared by a `HighCTX` element.

```
HighCTX IdContext is /* High Context*/
 HCTXState StateID: /*High States */
  BCTXStates : /*Basic Context States*/
  basic ContextID1 '/ ContextValue 'and
  basic ContextID2 '/ ContextValue '.
     Actions:
       /*Action declaration.*/.
     endact
  endHCTXSt
...
endHctx
```

CTXs-Maude grammar will be illustrated through the smart home system. The system is a kind of future home where pervasive computing emerges anywhere to facilitate our daily life. It is composed of two sub-systems, a fixed system and a Smart PDA (Personal Digital Assistant) or Smart phone. The fixed system is a kind of intelligent system responsible of monitoring and controlling home devices according to context variations. It is able to perform a universal remote control on all devices within the house. It can be used to lock, unlock all the doors and windows; it can be used to set the alarm automatically based on a schedule fixed on the smart PDA etc. Besides, the fixed system is permanently connected to the hand held smart PDA, allowing it to access smart house devices.

Many contexts can affect the smart house system execution based on location, user preferences, environment, time...etc. Using CTXs-Maude syntax, we specify SH-Context module. We consider the following declaration of a location context:

```
CTXmod SH-Context is
  Context HealthState is
   CTXDescrip: Monitoring Heartbeat, blood
     pressure…etc. levels of the body.
 CTXState:
  State:
    CTXValue -> AbnormalState.
    Actions:
       ExeAction invoke /IdInstance=
       wr_watch /Port= HS /Request:
       SendSignal AbNormal
    endact
   endState
       ...
endctxState
endctx
...
endctxm
```

We have declared here a basic context element `HealthState` that signals an abnormal state of the house owner, it is declared in the `CTXState` clause. The occurrence of such a state triggers execution of the associated action declared in the Actions clause.

We consider the scenario that the house owner is at a room entrance, detected via his PDA and location sensors, at the evening and it is dark, thus the room lights will turn on. They will turn off automatically when he leaves the room. This is expressed by the Auto-lighting high level context as follows:

```
HighCTX Auto-lighting is
 HCTXState Lighting:
   BCTXStates :
   Location / Room_Entrance and
   Environment / Dark.
    Actions:
      ExeAction invoke /IdInstance= smph
      /Port= PH /Request: Lightening Room.
    endact
   endHCTXSt
 endHctx
endctx
```

With the satisfaction of context values `Room_Entrance` for the Location context and dark for the Environment context, the smart home system executes a lightening action.

One pertinent clause in context declaration is the *Actions* one, since it specifies system reaction to context values changes. *Actions* clause is a parameter used to generate on the fly strategies to be performed on system structure and/or behavior. Two categories of actions are defined. The first one acts on system state by enforcing it to execute specific operations; and is declared using the keyword *ExeAction*.

```
ExeAction /*Action name.*/.
   /IdInstance=_ /*Instance identifier.*/.
   /Port=_ /*Port identifier.*/.
   /Request: _ /*Service call.*/.
```

The second category acts on system structure to modify its actual configuration; and is declared using keywords *CptAction*, *PrtAction…*, for adding new components instances, ports, connections, removing existing ones and so on. For adding a new component instance, the following syntax is used:

```
CptAction Actionid /*Action Name.*/.
  /Component (_)/*Component Type.*/
  Set IdInstance =_. /*Instance id.*/
```

For adding a new port, implying addition of new services, the following syntax is used:

```
PrtAction Actionid /*Action Name.*/
  /Port (_) /*Port identifier.*/.
  inComponent(_). /*Component Type.*/
```

Actions clause depends tightly on functional system authorized adaptations operations; each action declaration type corresponds to a structural/functional adaptation operation (*Actionid*).

Now we consider a scenario where the smart PDA and the fixed system periodically exchange information on Heartbeat, blood pressure, glucose levels and temperature levels of the body that are measured several times in the day via a wrist watch.

```
Actions:
  ExeAction invoke /IdInstance= Wr_Watch
  /Port= HS /Request:SendSignal(AbNormal).
endact
```

This ExeAction declaration corresponds to the wr_Watch unit reaction by invoking the SendSignal service; parameterized with abnormal state via its HS port (for Health State). In the same way, others actions can be declared and interpreted by the management layer to generate on the fly adaptation strategies.

4.3 Functional Layer:

System model is viewed as a set of components that provide system core functionalities. Functional components are defined and regrouped in CTXs-Maude in a component module as follows:

```
Cmod idMod is /*Component module .*/
  /* ports declaration.*/
  /* Components declaration.*/
  /* Architecture declaration.*/
Endcm
```

A component comprises a set of inner and outer ports. Each port contains a set of interfaces for services required or provided by the component. In CTXs-Maude, an input port specifies services provided by the component. The implementation of these services is defined in a Maude module attached to the port and implementing the corresponding services.

The various ports are considered bidirectional; the same port is used for sending requests and receiving responses. A connection is established between two instances of components whenever one component is providing the service and the other is requesting it. A component is defined by its inner and outer ports.

A configuration is an instance of an already defined architecture. It contains components instances that are created dynamically from components types declared in the architecture.

System behavior is ensured through two adaptation levels: adaptation actions and adaptation strategies. Adaptation actions have a specific realization to ensure that they are performed without affecting system consistency. They are invoked via specific adaptation strategies:

- Service execution: as invoke for invoking a component service on its corresponding port using the implementation module.
- configuration creation and adaptation : used on instances of the currently executing application by adding / removing items from the current configuration, as newconf for a new configuration creation, newIns, removePort newcnx…etc, for adding a new component instance, port instance, connection, or removing an existing one and so on. Such commands are directly executed on functional system architecture.
- Application update: for defining new versions of the considered application by adding new features or removing other ones using adaptation operations, that act on ports operation and implementation.

Since the proposed model defines context elements in an independent manner from system ones and only interactions between them are defined in the *Actions* clause of a context declaration, establishing such interactions, via on the fly adaptation strategies, is application independent. Hence the management layer of our model is generic enough and consists of a runtime environment implementing the generation and handling of on the fly strategies. This layer will be detailed in the following section.

4.4 Management Layer

The context layer obtains contextual information from different sensors and processes preliminary data filtering according to the context format (CTXs-Maude format) then sends it to the management layer. According to contextual information changes, Management layer generates adaptation rule to be applied on functional layer.

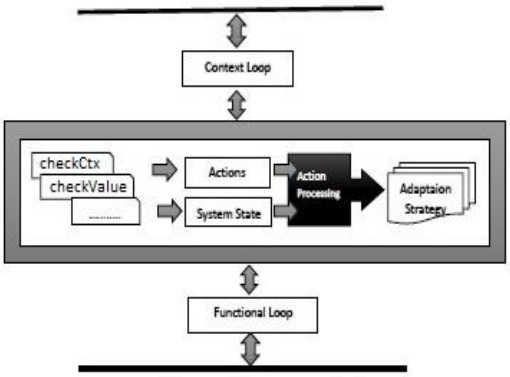

Figure 2. Context-aware system runtime environment

To ensure context-aware systems modeling requirements, namely context reusability and explicit representation of relationships with functional system, we declare and handle context and functional interfaces as two LOOP-MODE objects (see Fig 2).

The first Loop-Object, Context-Loop for instance, is the high level loop (supervisor) and manages input/output data to be exchanged with the user and/or the environment. Such data may contain contextual information or ordinary data for functional system. It has the following structure:

```
[ Input ,
< ContextModule;Functional-Loop > , Output]
```

- **Input:** input slot contains different data exchanged with the end user or the environment. Such data can be a new context value declared by a context command, a system

command or a system input data (system components specification module or context specification module…).

- **ContextModule:** Context specification, CTXs-MAUDE allows the specification of a context module with respect to the grammar already presented) for a full specification of contexts description, states and values, and actions declarations.

- **Functional-Loop**: functional system state is handled by the second Loop-object; called System-LOOP which stores functional system state and input/output data to be exchanged with the user. It has the following structure:

```
[Input ,
< SystemModule; Configuration > , Output]
```

With an input stream as first argument, an output stream as third argument, and a state as a second argument being composed of a functional system specification module and a system configuration composed of the actual components instances, connections…etc. System configuration has the following structure:

```
[ ConfigID, Instances, Connections ]
```

- **Output:** output slot displays different resulting data of actions, strategies execution, system evolution…etc.

The main role of the management layer is to handle the two Loop-Objects, and ensure inter-layers interaction via a set of operations to generate and execute on the fly strategies. Whenever some data is deposited in the input slot of the high level loop, it begins by analyzing it. If it is a new context value, the management environment triggers the associated action, specified in the context module, by generating the corresponding strategy. If it is an input data or a system command, it is directly forwarded to the input slot of Functional-Loop. The newly introduced data in Functional-Loop will be simply treated by the system itself.

Reconfiguration process is performed through two adaptation levels: adaptation actions and adaptation strategies. Adaptation actions have a specific realization to ensure that they are performed without affecting system consistency. They are invoked via specific adaptation strategies only. According to input values, different strategies may be triggered:

- Functional inputs:

Different functional inputs are treated; they are resumed in the following table:

Table 1. Functional inputs Type

Input Type	Example	Exe-Strategy
Functional command	'add 'instance	Specific strategies directly forwards System commands to the input slot of the Functional-Loop where they will be interpreted and executed.
Ordinary system data	Load a new Context specification module	Specific strategies perform specification module syntax analysis using the META-LEVEL operation meta-Parse.

- Contextual inputs :

Context command allows introducing a new low level context value. It has the following syntax:

```
Context: ContextID /Value: ContextValue .
```

As an example, `HealthState` and `Abnormal_State` are context identifier and the associated value respectively. The introduced data are firs deposited in the input slot of the Context-Loop object.

```
Context: HealthState /Value: Abnormal_State.
```

For a composed context state, the command has the following syntax:

```
HighContext:
basic ContextID1 / ContextValue and
basic ContextID2 / ContextValue
[ and ... . ]
```

As an example of a composite context introduction, we can have:

```
HighContext: Room_Entrance and
             Environment / Dark .
```

To achieve the required reconfiguration adaptation, and thus treat the newly introduced context value on the considered system, we first access to context declaration module to verify the existence of the context and its value. This is done by checkCtx operation. It takes as arguments the context command and context specification module (a CTXs-Maude module). If the context information (identifier) specified in the command exists in the context module, it calls `checkValue` operation which takes as arguments the context command and the state declaration to obtain the corresponding actions.

If the context information (value) specified in the command exists in the state declaration, it returns the corresponding actions declaration. The management layer will automatically generate on the fly specific strategies for this newly introduced context value. `ActionProcessing` operation is then responsible of executing adaptation actions on system structure and/or behavior.

`ActionProcessing` operation takes as arguments context actions declaration returned by `checkCtx` and the Functional-Loop. It recursively deposits adaptation commands in the input slot of the Functional-Loop using the META-LEVEL operation `metaXapply`. The later gets the meta-representation of the adaptation command and the actual Functional-Loop state (using `downTerm`, `upTerm` operations).

As an example of on the fly strategies generation and execution, we consider the situation when the wrist watch records any reading that is out of the threshold limits.

```
Context: HealthState /Value: Abnormal_State.
```

Context interpreter layer transmits this pair of values (context identifier and its value) to the Management layer which generates the corresponding on the fly adaptation strategy. The later consists of executing a `SendSignal` action, which sends a signal to the fixed system, and automatically alerts the user and/or searches for a nearby hospital using GPS to report that there is an emergency in need of an ambulance (see Fig 3).

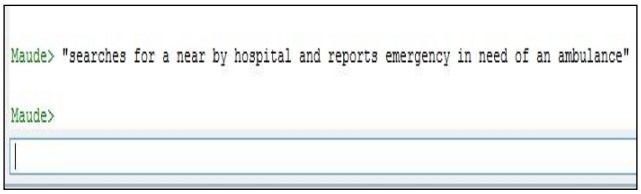

Figure 3. A Strategy application result.

As another example of strategies, we consider that user work office schedules an emergency meeting at night. In the morning, they check user schedule and find that (s)he has an open or free slot. By scheduling the early slot in user day, his Smart PDA asks the fixed system of the smart home to subsequently change user wakeup time but being still within the permitted boundaries. The fixed system is responsible of changing user wakeup time by modifying alarm settings.

```
Context: UserPreferences
 /Value: Scheduling_Event .
```

Then, the fixed system sets the alarm on based on the location where user sleeps.

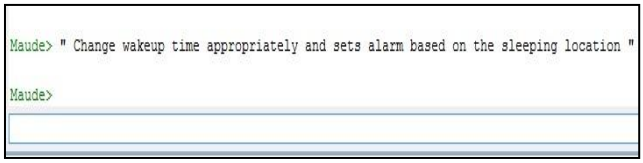

Figure 4. Maude console shows result.

5. CONCLUSION

We have proposed a formal model for context-aware adaptive systems specification. We have defined a layered model that includes a functional layer, a context layer and a management layer to control system behavior and/or structure in response to changes in the context. The management layer detects and checks changes in context values, and automatically generates specific adaptation strategies and triggers their execution on the functional system. The proposed model defines context model elements separated from-but related to System model through the management level. Furthermore, we have defined a Maude based language for the description of contexts and adaptation actions.

The separation of the context model from the system one reduces system modeling complexity and promotes maintainability of models where the two aspects; the system and its execution context, and their relationships can be clearly specified and managed. A direct consequence is the straightforwardness of reusing both system and context layers. The specification and representation of contexts and adaptation actions become easier using CTXs-Maude.

As future work, we intend to formally verify system adaptive behavior, and perform more validations on the approach applicability and practicality. Finally, and in the aim to facilitate the design and readability of context-aware systems an MDD approach will adopted to realize a framework for their specification assistance, running and verification.

6. REFERENCES

[1] Andrade, S.S., Macedo, R.J.A.M. 2010. Vers un cadre Conscient du contexte générique pour l'auto-adaptation des architectures orientées services. In the 9th International Conference on Internet and Web Applications and Services, pp. 181–184. Barcelona, Spain.

[2] Ayed, D., Delanote, D., Berbers, Y. 2007. MDD approach for the development of context-aware applications. In Proceedings of the 6th international and interdisciplinary conference on Modeling and using context, Springer-Verlag, Berlin, Heidelberg, 15-28.

[3] Bastide, G. 2008. Scorpio : A structural adaptation approach of software components application for ubiquitous environments. PhD. thesis, University of Nantes, France.

[4] Clavel M, Durán F., Eker S., Lincoln P, Martí-oliet N., Meseguer J., Talcott C. 2008 . Maude Manual (Version 2.4).

[5] Clavel M., Durán F., Eker S, Lincoln P., Martí-oliet N., Meseguer J., Talcott C. 2007. All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic. Lecture Notes in Computer Science 4350, Springer.

[6] D'Errico Liliana & Loreti Michele. 2011. Context-Aware Specification and Verification of Distributed Systems. In Roberto Bruni & Vladimiro Sassone, editors: TGC, Lecture Notes in Computer Science, vol. 7173, 142–159, Springer.

[7] Dhaussy P., Roger T. C., and Frédéric Boniol F. 2012. Context Aware Model-Checking for Embedded Software. In Embedded Systems - Theory and Design Methodology, 167–184.

[8] David, P.C. 2005. Développement de composants Fractal adaptatif: un langage dédié à l'aspect d'adaptation. Ph.D. thesis, University of Nantes, France.

[9] Gagnon P., Mokhati F.and Badri M. 2008. Applying Model Checking to Concurrent UML Models. Journal of Object Technology 7(1), 59–84.

[10] Gallardo M. D. M., Merino P., and Pimentel E. 2002. Debugging UML designs with model checking. In Journal of Object Technology, Vol.1(No. 2), 101–117.

[11] Garlan, D., Siewiorek, D.P., Smailagic, A., Steenkiste,P. 2004. Projet Aura: Towards Distraction-Free PervasiveComputing. IEEE Pervasive Computing. 1, 22–31.

[12] Garlan, D., Cheng, S.W., Huang, A.C., Schmerl, B., Steenkiste, P., 2000. Rainbow: architecture-based self-adaptation with reusable infrastructure. Computer. 37, 46-54.

[13] Gu, T., Pung, H. K., Zhang, D. Q. 2004. A middleware for building context-aware mobile services. In the 59th Vehicular Technology Conference, VTC 2004-Spring, 2656-2660. IEEE Press, New York.

[14] Martí-Oliet, N., Meseguer J., and Verdejo, A. 2005, Towards a Strategy Language for Maude. Electronic Notes in Theoretical Computer. Science 117, 417-441.

[15] Minh H., Tran, Colman A, Han J. and Zhang H.. 2012. Modeling and Verification of Context-Aware Systems. In Proceedings of the 2012 19th Asia-Pacific Software Engineering Conference - Volume 01, APSEC'12, IEEE Computer Society, Washington, DC, USA, 79–84.

Ranganathan, A., Campbell, R.H., 2008. Provably Correct Pervasive Computing Environments. In the IEEE International Conference on Pervasive Computing and Communications (PerCom), 160-169. 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications, pp. 71–74.