On Specifying and Verifying Context-aware Systems

Brahim Djoudi, Chafia Bouanaka and Nadia Zeghib
LIRE Laboratory, University of Constantine 2, Constantine, Algeria

Keywords: Context-Aware Adaptive Systems. Formal Methods. Meta-Programming. Model Checking. Maude.

Abstract: Software systems often need to be adapted for different execution environments, problem sets, and available resources to maintain and ensure their efficiency and reliability, being thus context-aware. Albeit, many approaches for context-aware systems specification have been proposed in the literature, the absence or poor representation of contextual information and its relationships with system entities without affecting system complexity and consistency usually leads to low-precision and irrelevant results. Moreover, it is difficult to verify the correctness of existing context models. In this paper, we propose a formal model for context-aware adaptive systems specification. The model also supports formal verification of the obtained system model through a set of inherent invariants, where context-aware systems behaviour can be verified according to system invariants by applying model checking techniques.

1 INTRODUCTION

Context-aware systems development and verification is a very complex task. A means to overcome such complexity is to adopt a formal support for modelling this category of applications, specifying adaptation scenarios and expressing global properties. A reasonable and desirable formal method (Gargantini et al, 2009) to use for this scope should be powerful enough to capture the principal models of computation and specification methods, and endowed with a meta-model-based definition conforming to the underlying meta-modelling framework. Additionally, it should be able to work at different levels of abstraction, and be executable, in order to validate meta-model semantics.

We adopt Maude (Clavel et al., 2008) as a formal semantic framework for the definition of a domain specific language for specifying and verifying context-aware systems. Our approach exploits mainly the reflection feature and meta-programming capability of Maude to enrich it with new constructors to obtain a domain specific language for context-aware systems specification, called CTXs-Maude (for **C**on**T**e**X**t-aware **s**ystems using **Maude**). The language grammar allows designers to specify context entities, system components and their relationships in terms of context states and actions to be performed whenever such states are reached.

The aim of this paper is to promote the ability to verify context-aware systems by proposing a formal model. In particular, our formal model establishes a clear separation of concerns between system and context entities. The goal is gained by the definition of a layered model where functional and context layers are designed in an entirely independent manner, only relationships or interactions between them are established via a set of dynamically generated adaptation strategies.

To establish interactions between functional and context layers, we propose a management layer which formulates context changes impact on system structure and behaviour. The main role of this layer is to interpret context changes to generate on the fly strategies to adapt system functionalities in terms of variations on system structure and/or behaviour.

To manage interactions between context and functional components, management layer manipulates two interfaces implemented as two Loop-Mode objects; one for functional system state and the other for context state. It also contains a set of operations to dynamically generate adaptation strategies and execute them.

Our proposed modelling methodology also allows verifying context-aware systems by presenting a systematic process for designing and verifying them.

The remainder of the paper is organized as follows: In section 2, we shows a motivating scenario using Adaptive Cruise Control (ACC)

Djoudi B., Bouanaka C. and Zeghib N..
On Specifying and Verifying Context-aware Systems.
DOI: 10.5220/0005000101810188
In *Proceedings of the 9th International Conference on Software Paradigm Trends* (ICSOFT-PT-2014), pages 181-188
ISBN: 978-989-758-037-6
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

system inspired from (Tran et al., 2012). Section 3 is dedicated to the presentation of our context-aware specification framework. Formal analysis of the ACC example in Section 4. Section 5 is evaluates the proposed model. A short conclusion and ongoing work round the paper up.

2 MOTIVATING SCENARIO

All concepts introduced in our model are illustrated through an example of a context-aware system, namely Adaptive Cruise Control (ACC) inspired from (Tran et al., 2012). The system consists of a set of electronic control units (ECUs) that are distributed and inter-connected over vehicle network (e.g., CANbus) to dynamically change driving conditions; e.g., adaptive cruise control, adaptive fuel management, adaptive suspensions,… etc. We will be more particularly interested with the Adaptive Cruise Control unit. The structure of an ACC consists of five software components modelling electronic automotive components: Controller Component, Engine Control Management Component (ECM), Radar Component, Weather Management Unit Component (WMU), and Road Management Unit Component (RMU).

The ACC unit main role is to control vehicle speed based on driver pre-set parameters. It is able to adjust it to maintain a safety time gap with a preceding vehicle, called target. Thus, the ACC behaviour is tightly dependent on vehicle speed and distance from the preceding vehicle. It can be activated in specific contexts only as target detection by the radar component. In such situation, the ACC calculates a decelerating distance to match target speed and maintain the suitable distance. Then, it enforces vehicle speed deceleration until a matching point; the relative speed between the two vehicles becomes zero, is reached.

3 CONTEXT-AWARE SYSTEMS SPECIFICATION

The formal model proposed here for context-aware adaptive software systems specification and verification focuses on context specification and system/context interactions modelling with respect to system consistency and safety.

The model is composed of four layers. The top layer is a sensing one that collects contextual information using different sensor types (physical sensors, virtual sensors, logical sensors). Then, the context layer operates preliminary data filtering and interpretation of contextual information. The basic layer is a functional layer providing system core functionalities. Dynamic interaction between functional and context layers is established via the management layer guarantying dynamic adaptation in a transparent manner.

3.1 Context Elements Specification

Context model serves to specify contextual entities relevant to system operation and/or adaptation. A concrete context is defined as any information characterized by particular states; considered to be relevant to user interaction with the application and having an impact on system structure and/or behaviour. A context element is identified by its identity, the sensor provider URL (context sources ID), and context elements states that specify context possible values and the corresponding system adaptation. Each context state is defined by a pair of attributes: a context value and a set of actions to be performed on system structure and/or state whenever the context value is reached. Hence, relationship with functional system is explicitly and clearly specified via Actions attribute of a context element.

Since Maude (Clavel et al., 2008) allows specifying modules with user-definable syntax by exploiting its reflection and meta-programming properties, we define a domain specific language; CTXs-Maude, on top of core Maude introducing new constructors to allow specifying context-aware adaptive systems. CTXs-Maude grammar will be illustrated through the adaptive cruise control (ACC) system.

A context module in CTXs-Maude includes a set of context descriptions. It has the following syntax:

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

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

```
Context IdContext is
  CTXSensor: /*Sensor URL.*/
    CTXState: /*Context States*/

      State:
        CTXValue -> /*Context Value.*/.
        Actions:
           /*Action declaration.*/.
        endact
      endState
  ...
 endctxState
endctx
```

Different relevant contexts affecting ACC system behaviours can be identified. The ACC controller behaviour tightly depends on the following context elements:

- Travelling conditions changes (bends, whether...),
- Vehicle speed (faster than a maximum speed , say 100 km/h),
- A safety time gap, a minimum safety time gap is of 2 seconds, with a preceding vehicle.

The ACC reaction to context fluctuations includes three main states: closing zone, coasting zone and matching zone. Closing zone denotes a state where the vehicle detects a target and starts calculating a decelerating distance. Coasting, zone denotes a state where the vehicle starts decelerating. Matching zone is gained whenever the relative speed between the two vehicles becomes zero, i.e., default time gap is of 2 seconds.

For example, when a vehicle travels in a rainy condition or through sharp bends, the ACC unit automatically reduces its speed. It resumes the initial settings when driving conditions become safe (e.g., no rain or sharp turns). Using CTXs-Maude syntax, the ACC-Context is specified in the following module:

```
CTXmod ACC-Context is
 CTXSensor: WMU.
  CTXState:
  State :
     CTXValue  ->  Rainy   .
   Actions:
    ExeAction invoke
      /IdInstance= CTRL
      /Port= PS
       /Request: ReduceSpeed(Rainy).
   endact
  endState

  State :
   CTXValue  ->  Fine   .
     Actions:
   ExeAction invoke
    /IdInstance= ctrl
    /Port= PS
    /Requests: PresetSpeed(Fine).
   endact
  endState
   ...
 endctxState
 endctx
```

One pertinent clause in a context declaration is the *Actions* one, since it specifies system reaction to context values changes. Two categories of actions are defined. The first category 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; it uses keywords CptAction, PrtAction...; for adding new instances, ports, connections, removing an existing one and so on. For adding a new instance, the following syntax is used:

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

Actions clause depends tightly on functional system adaptation strategies. Each action declaration type corresponds to a structural/functional adaptation (`Actionid`) strategy, to be presented in the next section. As an example of actions, a target vehicle detection by the radar component implies the execution a *CalculSpeed* service:

```
Actions :
    ExeAction invoke
    /IdInstance = CTRL  /Port = PS
    /Request: CalculSpeed (Closing).
 endact
```

The above `ExeAction` declaration corresponds to the ACC unit reaction. It consists of invoking the `CalculSpeed` service which is parameterized with closing state, on CTRL component via its PS port. In the same way, other actions and strategies are used and interpreted by the management layer to generate on the fly adaptation strategies.

Sometimes, system functionalities and adaptation actions execution are triggered by a conjunction of two or more different contexts. This situation is defined in CTXs-Maude via high level or composed contexts.

```
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
```

An example of a high level context represents the controller component reactivation whenever the minimum safety requirements are reached in despite of a break being already applied by the driver. Such situation is specified by a high level context as follows:

```
HighCTX  VitalSafety is
  HCTXState matchSpeed   :
   BCTXStates :
   Radar / Target-Detected and
   VehiclesSpeed / >>Radar.TargetSpeed
   and TimeGap / 2seconds .
   Actions:
      CptAction resume  /Component
     (Controller)set IdInstance = ctr
      ExeAction invoke
      /IdInstance= CRTL /Port= PS
      /Request: ReduceSpeed (Matching).
   endact
  endHCTXSt
 endHctx
```

VitalSafety context expresses the fact that, even though the driver has applied a break to take full control on its vehicle, if the radar detects a preceding vehicle with a running speed inferior of the actual speed of the vehicle, the Controller unit might be reactivated to reduce vehicle speed. After achieving a minimum time gap of 2 seconds, the Controller is automatically reactivated. Such high level context is declared in CTXs-Maude by specifying two distinct actions. The first one reactivates the Controller if it is deactivated by executing the *resume* functional strategy responsible of activating the *CRTL* blocked component. The second action matches the vehicle speed by invoking the *ReduceSpeed (Matching)* service on the resumed component.

The proposed syntax of context components allows declaring and handling most of anticipated and unanticipated changes; giving the user a large flexibility in specifying his application context and the corresponding system reaction.

3.2 Functional Elements Specification

The system model is viewed as a set of components that provide system core functionalities. Functional components are defined and grouped 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 operations 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 a predefined architecture, containing components instances that are created dynamically from components types already declared in the architecture.

3.3 Managing Context/System Interactions

Management layer principal role consists of interpreting and using context changes to generate adaptation strategies to adapt system functionalities in terms of variations on application structure and/or behaviour.

To ensure context/system independence, a strict separation of concerns is adopted. The only way to establish interaction channels between instances of functional and context models are realized via dynamically generated adaptation strategies. These strategies are parameterized with actions declared in the context module declared with respect to CTXs-Maude. Thus, a generic format of on the fly strategies adaptation strategies is defined.

Management layer is the core layer of our framework. It is responsible for parsing and interpreting context values and forwarding their impacts on system structure and behaviour. Management layer is composed of two interfaces (see Figure 1), one for context input/output and the other for functional data acquisition to capture system current state in order to perform the suitable adaptation actions if needed, and a set of operations and rules responsible of generating and executing adaptation strategies.

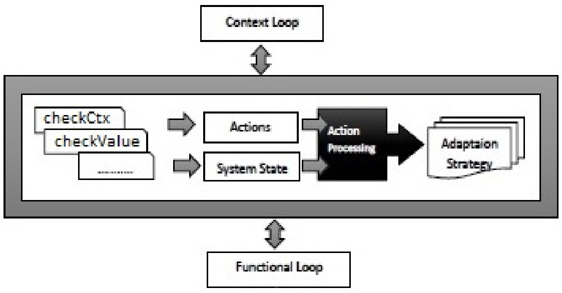

Figure 1: Management layer structure.

Context and/or system state changes are used as triggers to evaluate conditions of adaptation. Management layer uses specific adaptation strategies to establish relations between context level and functional level in an abstract manner. On the fly strategies are generated automatically from interpreting context actions which represent system reaction to contextual change and are applied on functional system state.

To achieve the required reconfiguration on the considered system according to context changes, management layer first accesses to context declaration module to obtain the set of adaptation actions to be performed on system configuration. `checkCtx` and `checkValue` operations take as arguments context tuple and context specification module (CTXs-Maude module) and check if the newly introduced context identifier and the corresponding context value exist in the context module declaration. If the verification succeeds, the corresponding context actions declarations of context state clause are returned.

After interpreting context actions and based on system state, an on the fly adaptation strategy to be applied on functional system state is generated. This is the role of `ActionProcessing` action. It takes as arguments context actions declaration and system state. It recursively applies adaptation actions on the current system state to accomplish the desired adaptation operations. The newly obtained system state and outputs are raised to the Context-Loop and placed in the corresponding slots.

As an example of on the fly strategies generation and execution, we consider the situation where the vehicle is actually traveling on sharp bends. As soon as sensing layer detects a road shape modification (bends), context interpreter layer transmits a pair of values, context identifier and its value, to the management layer in the following format:

```
'Context:'RoadStat '/Value:'Sharp-bends
```

The management layer generates the corresponding on the fly strategy that invokes the ACC `ReduceSpeed` service. The later systematically reduces vehicle speed (see Figure 2).

```
Maude>...
Maude> Start vehicle decelerating to
       avoid vehicle deviation
```

Figure 2: A strategy application result.

Whenever a break is applied, the ACC component might be disengaged regardless its current state allowing the driver to take full control on its vehicle. However, the controller might be reactivated automatically if *VitalSafety* context is reached to maintain safety purposes (see Figure 3).

```
Maude> Instance ctrl Stopped
Maude> ...
Maude> Instance ctrl Resumed
Maude> ctrl maintain save relative
       speed between vehicles
```

Figure 3: CTRL component deactivation and resume strategies.

4 CONTEXT-AWARE SYSTEMS VERIFICATION

Model Checking (Gagnon et al, 2008) is a formal verification technique to be applied on a system abstract model to determine whether a series of properties are satisfied by the considered system. According to Gallardo & al (Gallardo and al, 2002), model checking is one of the most useful results of research in formal methods to increase software quality. A model checker is an automatic tool that confronts two descriptions of system behaviour, one being considered as the required behaviour and the other the actual design (Gallardo et al, 2002). The main usefulness of such a technique is the fact that the automatic tool, upon encountering an error state, returns a counterexample illustrating the path taken to reach that state.

In the present work, we deal with reachability, safety and liveness properties verification through the modelling of the ACC system.

Intuitively, reachability property verifies whether a certain system state is reachable from a given initial state. Safety properties (Tran et al., 2012) ensure that nothing bad will ever occur, whereas liveness properties stipulate that something good will eventually happen. The Maude search command is used to check that our ACC formal model satisfies the considered properties, or violates them by furnishing a useful counterexample.

The Maude search command (Clavel et al, 2008) allows exploring system state space, following a breadth-first strategy in different ways, to verify whether the given property is violated or not. The model checking result is either no state violates the considered invariant or a state violating it together with the sequence of rewrites being executed from the initial state to attain such state that is a counterexample. The search command syntax conforms to the following general scheme:

```
search in (ModId ) :
      (Term-1) (SearchArrow) (Term-2)
      such that (Condition)
   .
```

Where:

- The module ModId specifies where the search command takes place. It can be omitted;
- Term-1 is the starting term;
- Term-2 is the pattern that has to be reached;
- SearchArrow is an arrow indicating the form of the rewriting proof from Term-1 until Term-2. There are different arrow forms :

  – =>1 means a rewriting proof consisting of exactly one step,
  – =>+ means a rewriting proof consisting of one or more steps,
  – =>* means a proof consisting of none, one, or more steps, and
  – =>! Indicates that only canonical final states are allowed, that is, states that cannot be further rewritten.
- Condition specifies an optional property that has to be satisfied by the reached state; the syntactic form of the condition is the same as the one of conditions for conditional equations and memberships.

The key question in the ACC system is whether the adaptive behaviour can be applied but still maintaining critical safety requirements. We are interested here with the following invariants.

a. The controller disengages whenever the driver applies a Break Event (safety).
b. If a preceding car is detected, the controller cannot react since it is deactivated; otherwise the controller reduces the car speed (liveness).
c. After an Apply-break event, the critical safety state can always be reached and the controller might be reactivated automatically (reachability property).

To achieve the required verification on the considered system according to context changes, our runtime environment implements context changes handling in a module called CTXMod.maude providing a set of context adaptation strategies to be performed on system configuration according to contextual situations. One initial state corresponds to the state where the driving conditions are suitable as straight path road. We can now verify the reachability of a state where the controller disengages after a user apply break.

By executing the search command (see Figure 4), Maude model checker finds three states where the controller is always blocked after an apply break. Thus, invariant (a) is verified.

```
Maude>  search in CTXMod:
['Context:'RoudStat'/Value:'Straight-path
,< ContextModule; SysConfig >,
Output] =>*[CTX,< CtxModule ; SysConfig ,
Output] such that
CTX == 'Context:'UserPrefrence
      '/Value: 'ApplyBreak '.
And
SysConfig ==...Ctrl 'Controller BLOKED…
states: 3  rewrites: 70 in 2675208203ms
cpu (716ms real) (0 rewrites/second)
Solution 1 (state 2)
states: 3  rewrites: 79 in 2675208203ms
cpu (33ms real) (0 rewrites/second)

CTX --> 'Context: 'UserPrefrence '/Value:
'ApplyBreak '.
...
LI -->  'ecm 'ECM EXECUTED..., 'Ctrl
       'Controller BLOKED...
No more solutions.
```

Figure 4: Maude console shows verification result.

However, if a preceding car is detected, the controller cannot react since it is deactivated.

To verify that our model satisfies the invariant (b) (see Figure 5), we attempt to verify the negation of such invariant. From an initial state where the controller is blocked due to a break event, is it possible to attain a state where the controller is activated and controls the car speed.

```
Maude>...
Maude>  search in CTXMod:
['Context:'UserPrefrence'/Value:
'ApplyBreak'., <ContextModule;
...Ctrl 'Controller BLOKED ...>,
Output]=>* [CTX,< CtxModule; SysConfig ,
Output] such that
CTX = 'Context: 'Radar
      '/Value 'Target-Detected '.
And
SysConfig=...Ctrl 'Controller   EXECUTED
...
No solution.
Maude>...
```

Figure 5: Maude console shows verification result.

No solution is found by the search command. Thus, there are no states violating the negation of our invariant.

To ensure safety property or invariant (c), the controller might be reactivated automatically when critical safety requirements are in place. This invariant is verified by the next search command.

Table 1: Comparison of current context-aware adaptive systems.

	Reuse	Context Modelling	System-Context Relationships	Properties Verification	User Conformance
AURA (Garlan et al., 2002)	NS	PS	NS	NS	NS
SOCAM (Gu et al., 2004)	NS	PS	NS	NS	NS
MDD (Ayed et al., 2007)	PS	S	S	PS	S
Tran et al. (Tran et al., 2012)	PS	PS	PS	S	PS
Ranganathan and Campbell (Ranganathan and Campbell, 2008)	PS	PS	PS	S	PS
Dhaussy et al. (Dhaussy et al. ,2012)	PS	PS	PS	PS	NS
Our Approach	S	S	S	S	S

```
Maude>...
Maude>  search in CTXMod:
['Context:'UserPrefrence'/Value:
'ApplyBreak'., <ContextModule;
...Ctrl 'Controller BLOKED ...>,
Output]=>* [CTX,< CtxModule; SysConfig ,
Output] such that
CTX = 'Context: 'Radar
      '/Value 'Target-Detected '.
And
SysConfig=...Ctrl 'Controller   EXECUTED
...
No solution.
Maude>...
```

Figure 6: Critical Safety property verification.

5 DISCUSSION

Compared to existing approaches, our model has the following advantages. Primarily, the proposed framework design can perform adaptive behaviour for context-aware systems but still maintaining system invariant. The separation of concerns between context model elements and system ones reduces the design complexity and increases model reusability and maintainability. The introduced CTX-Maude specification language supports hierarchical structures modelling by composing both context elements and functional components.

The Management runtime layer allows users to select adaptation actions to be applied without caring about how these actions are implemented and executed. Therefore, the software system can perceive execution context changes and make the adequate actions in a transparent manner without any user attention. This feature gives users great facility and flexibility when using CTXs-Maude.

Additionally, model checking techniques provide a very good guide of system design correcteness. The experimental results of Adaptive Cruise Control safety critical requirements verification reflect that the proposed model ensure system invariant.

Table 1 resumes a comparative study that we have realized on some significant works on context-aware pervasive systems specification and verification and highlights our contribution maincharacteristics, where S means that the requirement is satisfied, NS for non-satisfied requirement and PS for partially satisfied requirement.

6 CONCLUSION

In this paper, we have proposed a generic formal model for context-aware adaptive systems specification and verification that establishes a clear separation of concerns between system entities and context ones. A layered model is defined where functional system and context layers are designed in an entirely independent manner. Only relationships between the two layers are established via a core layer. The later is responsible of specifying context changes impact on system functionalities in terms of variations on application structure and behaviour, by applying different reconfiguration strategies. These strategies are parameterized by context values changes and the associated actions on system structure, giving the Management layer a high level of genericity and reusability.

We have adopted Maude as a semantic framework for the proposed model and exploited its reflection and meta-programming capabilities to enrich it with context-awareness concepts. We have also implemented our model by proposing a runtime environment for context-aware systems execution. The runtime environment exploits two Loop-Mode objects to manage context and system states. Interactions between the two loops are realized via ascend and descent operations and strategies.

Model validation is realized via the formal verification of reachability, safety and liveness properties of a concrete system, the ACC controller for instance. A set of formal properties expressing reachability, safety and liveness requirements have been defined and verified using Maude model checker.

As future work, we intend to complement our framework for context-aware systems development with necessary editors for designing, executing and verifying the considered systems. We aim to facilitate the design and readability of systems by associating graphical representations to the algebraic specifications defined by CTXs-Maude.

REFERENCES

Gargantini, A., Riccobene, E., Scandurra, P., 2009. 'A semantic framework for metamodel-based languages.', *Autom. Softw. Eng.* Vol 16, pp. 415-454.

Clavel, M., Durán, F., Eker, S., Lincoln, P., Marti-Oliet, N., Meseguer, J. & Talcott, C., 2008. 'Maude Manual (version 2.4).

Tran, M. H., Colman, A. W., Han, J. & Zhang, H., 2012. Modeling and Verification of Context-Aware Systems. *in*. 'APSEC', IEEE, pp. 79-84.

Gagnon, P., Mokhati, F., Badri, M., 2008. 'Applying Model Checking to Concurrent UML Models.' *Journal of Object Technology* , Vol 7, pp. 59-84.

del Mar Gallardo, M., Merino, P. & Pimentel, E. (2002), 'Debugging UML Designs with Model Checking.', *Journal of Object Technology***,** Vol 1, pp. 101-117 .

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

Gu, T.; Pung, H. K. & Zhang, D. Q. (2004), A middleware for building context-aware mobile services, *in* 'Vehicular Technology Conference, 2004. VTC 2004-Spring. . IEEE Press, Vol 5, pp. 2656-2660.

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, pp.15-28.

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

Ranganathan, A., Campbell, R. H., 2008. Provably Correct Pervasive Computing Environments, *in* 'PerCom', IEEE Computer Society, pp. 160-169.