CAMidO, A Context-Aware Middleware based on Ontology meta-model

Nabiha Belhanafi, Chantal Taconet and Guy Bernard

GET / INT, CNRS Samovar 5157,
9 rue Charles Fourier, 91011 Évry, France,
{ Nabiha.Belhanafi, Chantal.Taconet, Guy.Bernard}@int-evry.fr

Abstract. With the expansion of wireless communication and mobile hand-held devices, computing becomes increasingly mobile and pervasive, this mobility and the characteristics of those devices lead to the creation of location- and context-aware applications. Most of research work dealing with context and adaptation focus their effort either in proposing context-models for context description (application developer has to deal with context and adaptation), or in proposing platforms that interact with context and adapt applications to context changes without providing any meta-model for context description. It lacks a proposition which includes both a model and a middleware. This paper describes CAMidO, a Context-Aware Middleware based on Ontology meta-model that provides a meta-model for context description. This description is used by the CAMidO middleware for context collection and applications reactions to significant context changes. The aim of such a middleware is to help developers to create context-aware applications.

1 Introduction

The recent advance in wireless networking technologies and mobile computing reports changes in applications execution environment which becomes dynamic. Thus it becomes necessary that applications get aware of environment changes and adapt their behaviour accordingly.

To achieve such level of awareness, one solution is to let application designers describe context to which each application is sensitive, collect context information from associated sensors, detect context changes and adapt the application to those changes. As these are complex tasks for the developer, a better solution is to use a middleware upon the network operating system. CAMidO, the middleware we present in this paper is one of those middlewares. The particularities of CAMidO is to provide a meta-model for context description and to automate all tasks dealing with context and adaptation thanks to code generation and rules reasoning.

In this paper, we describe the CAMidO middleware and its meta-model. Section 2 presents our motivations and objectives. Section 3 describes the proposed meta-model and gives an example scenario to illustrate how it can be used to create a context-aware application. The CAMidO middleware is presented in Section 4. Section 5 discusses some related works and Section 6 concludes the paper.

2 Motivations and objectives

Component-based middleware such as CORBA Component Model [1] and Fractal [2], relieve application developers from a number of painful tasks. These middleware use description of non functional properties and of service invocations to generate automatically the appropriate source code dealing with the described properties. With CAMidO, we apply component non functional properties to context-aware properties. The idea is to ease development of context-aware applications by providing context-meta model for context description. These descriptions generate automatically the appropriate source code to deal with context collection and application reactions to context changes.

A lot of research works focus their efforts either in proposing context-models which ease creation of context-aware applications or in proposing platforms for context management and application adaptation, whereas few research efforts have been undertaken to propose context-model and general mechanism for context management to be integrated to component-based middleware. The proposed middleware provides a meta-model for context description and contains context-aware components which allow context management and code generation in order to apply application adaptation.

The existing context models use conceptual modelling approaches to represent context information such as Entity/Relation model, class diagram UML or CML [3], others use ontology for context modelling. The proposed meta-model is based on an Ontology description, because it allows data distribution, information reasoning and knowledge reuse. The proposed ontology (cf . Section 3.1), gathers information which can be reused by all context-aware applications such as sensor and context information. This ontology provides the middleware information about context interpretation, and reactions to be triggered if a relevant context is detected.

Modelling sensors information gives the context-aware infrastructure the possibility to interact easily with new sensors, and the application designers more flexibility when creating context-aware applications. Indeed, sensors are integrated to the middleware without compilation and the designer has the possibility to choose different sensors to collect the same type of context information.

3 Context Modelling

The purpose of this section is to describe the proposed ontology-based meta-model for context-aware applications and services.

Ontology is used to specify the conceptualisation of an application by describing relationships between application's entities. It enables knowledge sharing, distribution and reuse of defined concepts without starting from scratch.

The proposed meta-model takes benefit from these characteristics for modelling context-aware applications and distributing context information between components of an application. This meta-model supports semantic context-aware representation by defining the upper common concepts for context-aware applications in general, and by describing properties and relationships between those concepts.

To illustrate the use of the proposed meta-model for the creation of context-aware applications, we give in this section a simple example for conference room management. This room is equipped with indoor location sensors. Each participant wears an

Active Badge [4], this device detects its indoor location. Network Badge Sensors are placed in the conference room to enable capturing signals emitted by each Active Badge. A master server connected to the network makes location information at the disposal of the middleware. When a person enters in the conference room, its presence is detected, then the conference program and the presented papers are automatically transfered to its laptop or PDA. During the conference, when a paper is presented by its author, it is automatically opened in the PDA or the laptop of each person present in the conference room.

The CAMidO meta-model is described in details in Section 3.1, and the modelling of the example scenario is given in Section 3.2.

3.1 CAMidO meta-model

In this section we describe the CAMidO meta-model for context-aware services and applications. This meta-model allows designers to create context-aware applications and reuse defined vocabularies and context.

The proposed meta-model is based on an ontology description, this ontology is written in the OWL language [5]. It allows description of context, sensors from which data are collected, interpretation rules and adaptation policies. Two types of adaptation are taken into account, reactive and proactive adaptations. Reactive adaptation consists in listening to changes in the execution context and reacting accordingly. Whereas proactive adaptation, which concerns service request, consists in anticipating the way in which the service is provided by changing the source code of the service policy by an other policy (the same service can be provided using different policies) according to context changes.

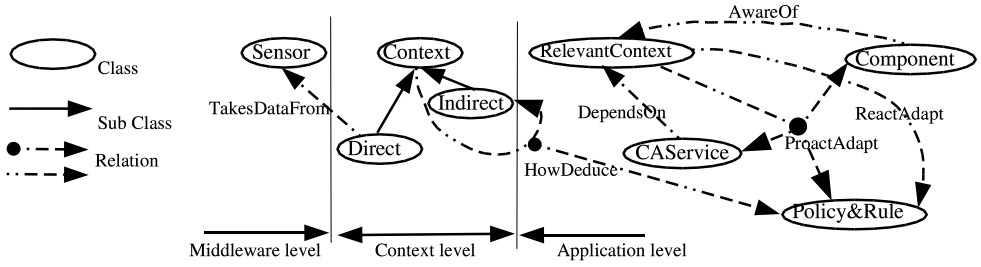

Fig. 1. The CAMido meta-model

As it is illustrated in Figure 1, the proposed meta-model is divided in three levels, an ontology is associated to each level.

The first ontology is associated to the middleware level, it concerns sensors description. This ontology contains information about sensors that the middleware can interact with. This information is created and updated easily by the middleware maintenance

agent to enable the middleware to interact with new sensors. Sensors can be used by all applications running on top of this middleware.

To enable the middleware to capture location information from Active Badge, the sensor instance (ActiveBadge) illustrated in Figure 2 has to be described in the ontology. The Active Badge sensor instance allows the middleware to interact with the master server, in order to detect the presence of a person in the conference room and give its geographic coordinates (*x*,*y*).

```
<Sensor rdf:ID="ActivBadge">
  <SensorName>ActivBadgeSensor</SensorName>
  <SensorSourcePath>http://www-inf.int-evry.fr/~belhanaf/Sensor/ActiveBadge</SensorSourcePath>
  <SensorDataParam>X#Y</SensorDataParam>
  <SensorParamType>Integer#Integer</SensorParamType>
</Sensor>
```

Fig. 2. Description of the Active Badge sensor

The second ontology of Figure 1 is associated to the context level. It gathers information about context to which all context-aware applications, described using CAMidO meta-model, are sensitive. The *Context* class belongs to this ontology. The direct context is captured directly from sensors, and the indirect context is interpreted from other contexts.

The example scenario is sensitive about *PersonLocation* which is captured directly from the Active Badge sensor and *ProximityToMicrophone* which is interpreted from *PersonLocation*. These two contexts are described in this ontology as illustrated in Figure 3. Interpretation policy belongs to the application level, its description is done in the third ontology.

```
<owl:Class rdf:ID="ProximityToMicrophone">
  <rdfs:subClassOf rdf:resource="#Indirect"/>
</owl:Class>
<owl:Class rdf:ID="PersonLocation">
  <rdfs:subClassOf rdf:resource="#Direct"/>
  <TakesDataFrom rdf:resource="#ActivBadge"/>
</owl:Class>
```

Fig. 3. Example of Context Description

The third ontology is associated to the application level, it gathers information specific to the application allowing the designer to describe:

- All relevant context to which the application is sensitive by creating instances of the *RelevantContext* class.
- Context-aware components belonging to the application by creating instances of the *Component* class, and binding them with the described relevant context using the *AwareOf* property.

- Interpretation rules for indirect context description, by creating instances of the *Policy&Rule* class to describe interpretation methods and the *HowDeduce* property which enables to specify the indirect context to be deduced using context information.
- Reactive adaptation of the application, when a relevant context is detected, by describing the *ReactAdapt* property which binds relevant context with the associated adaptation method described in the *Policy&Rule* class.
- Proactive adaptation of the application, by describing the *ProactAdapt* property which binds relevant context with the policy to be invoked (described as instance of the *Policy&Rule* class), and the component (belonging to the *Component* class) which invokes the service described in the *CAService* class.
- Context-aware services installed on top of the CAMidO middleware [6] and context to which they are sensitive, by creating instances of the *CAService* class and binding each instance to the relevant context to which it is sensitive using the *DependsOn* property.

These informations are used by CAMidO to manage context, interact with sensors and provide adaptation mechanism for applications.

3.2 Example Scenario

The conference room management application is made up of three components. The *PaperRepository* component is installed in the server, it contains all papers to be presented during the conference. The *PaperDownloader* component is installed in the PDA (Personal Digital Assistant) of each participant, it enables to download the conference program and all the papers of the conference. The *DisplayerAgent* Component is installed in the PDA of each participant and enables to display a selected paper in the PDA.

The conference room application is sensitive about location and proximity. The latest context is deduced from *PersonLocation*. Figure 4 describes interpretation rules which enable to interpret *ProximityToMicrophone*. This context is deduced by comparing the person position with the microphone position using the *NearMicrophone* method.

```
<HowDeduce rdf:ID="deduceProximity">
  <HowDeduceDomaine>PersonLocation</HowDeduceDomaine>
  <HowDeduceRange>ProximityToMicrophone</HowDeduceRange>
  <ExecuteDeduction rdf:resource="#nearMicrophone"/>
</HowDeduce>
```

Fig. 4. Example of Indirect Context Interpretation

Figure 5 illustrates description of the *NearMicrophone* method as instance of the *Policy&Rule* class by specifying the path of the class containing the method source code, its name, the type of its parameters, and the type of the returned value. This

method enables the middleware to determine whether a person location expressed by its *personX* and *personY* coordinates is near the microphone located in the conference room with its *microphoneX* and *microphoneY* coordinates.

```
< Policy&Rule rdf:ID="NearMicrophone">
      <NameAction>NearMicrophone</NameAction>
      <ParamType>varInteger#varInteger#Integer#Integer</ParamType>
      <ReturnedType>String<ReturnedType>
      <Path>....</Path>
</Policy&Rule>
```

Fig. 5. Example of Interpretation method

Figure 6 describes the relevant context and the adaptation policy which has to be invoked when this context is detected. This policy consists in opening the speaker's paper when he is near the microphone. This method has to be described as an instance of the *Policy&Rule* class.

```
<RelevantContext rdf:ID="relevantProximityToMicrophone">
  <ContextName>ProximityToMicrophone</ContextName>
  <Operation>Equal</Operation>
  <Value>Near</Value>
</RelevantContext>
<HowAdapt rdf:ID="adaptProximityToMicrophone">
  <IfRelevantContext rdf:resource="#relevantProximityToMicrophone"/>
  <ExecuteAdaptation rdf:resource="#openSpeakerPaper"/>
</HowAdapt>
```

Fig. 6. Example of Adaptation policy

Each component of the application is sensitive about a particular relevant context as illustrated in Figure 7. The *PaperDownloader* component is sensitive about person location, whereas the *DisplayerAgent* component is sensitive about proximity.

```
<Component rdf:ID="PaperDownloader">
  <ComponentName>PaperDownloader</ComponentName>
  <AwareOf rdf:resource="#relevantInConferenceRoom"/>
</Component>
<Component rdf:ID="DisplayerAgent">
  <ComponentName>DisplayerAgent</ComponentName>
  <AwareOf rdf:resource="#relevantProximityToMicrophone"/>
</Component>
```

Fig. 7. Example of context-aware component description

All these descriptions are used by the context-aware middleware presented in Section 4 to manage context information, detect relevant context changes and trigger adaptation policies.

4 CAMidO Architecture

CAMidO middleware is a component oriented middleware which provides an ontology-based meta-model for creating context-aware applications, it automates context management by adding context-aware components to the middleware and uses container characteristics to automate adaptation process as it is described in this section.

In order to adapt an application to context changes, we follow four main steps: (1) the collection of context information, (2) the interpretation of the collected data, (3) the analysis of those data and decision making about application reactions, (4) the triggering of the decided reactions.

All these steps are carried out by CAMidO which provides the proposed meta-model for context description. The described information is used for managing context, interacting with sensors and adapting the application to context changes. As shown in

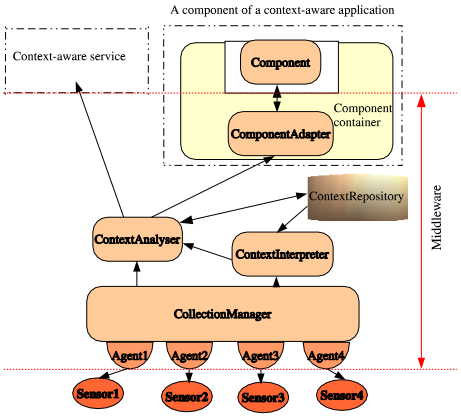

Fig. 8. CAMidO Architecture Overview

Figure 8, this middleware is made up of five components.

The *CollectionManager* is in charge of collecting context information from sensors according to designer description. For each sensor an agent which is an entity allowing interaction with a sensor type is activated for collecting data from it. Each agent knows how to interact with the associated sensor by using the described information in the *Sensor* class. The collected data are transfered by the *CollectionManager* to the *ContextAnalyser* and the *ContextInterpreter*.

The *ContextAnalyser* is responsible for filtering context information and determining relevant changes. Context filtering consists in detecting context changes by comparing

the collected context value with its old value stored in the *ContextRepository*. If a relevant context occurs, the new context value is saved in the *ContextRepository* by the *ContextAnalyser* which notifies the subscribed component about this relevant context changes.
The *ContextInterpreter* has to deduce high level context information by using the *HowDeduce* relation provided by CAMidO meta-model.
The *ContextRepository* stores context information.
The *ComponentAdapter* has to adapt an application component to context changes according to the adaptation rules defined by the application designer. It subscribes to the *ContextAnalyser* for relevant context in order to be notified when it occurs for applying adaptation policies. The *ComponentAdapter* belongs to the component container [7], it is made of a lot of components working together in order to adapt application's component to context changes.

If a component belonging to an application is sensitive to a given context, it subscribes itself to the *ContextAnalyser* in order to be notified when relevant changes occurs. The *CollectionManager* activates agents which are not activated yet, responsible for collecting context information, then the following process is executed for each context type: (1) each agent collects context information, (2) the *CollectionManager* sends the collected value to the *ContextAnalyser* and the *ContextInterpreter*, (3) the *ContextInterpreter* deduce high level context information using interpretation rules and transfers the interpreted data to the *ContextAnalyser*, (4) the *ContextAnalyser* filters the sensed and the interpreted data to detect relevant context changes, then notifies the subscribed component, and the *ContextAnalyser* saves the new context value in the *ContextRepository*, (5) go to step 1.

The described informations are used by the CAMidO compiler to generate automatically source code for application adaptation and Rule files, to be used by the *ContextAnalyser* and the *ContextInterpreter*, for relevant context changes detection.

Adaptation which consists in reactive and proactive adaptation is performed using the component container [7] because it controls all component interactions through entities called *controller*. In most of the component models, containers offer at least two controllers. The first one intercepts all the incoming request (*pre-request interceptor*) whereas the second one intercepts all the outgoing requests (*post-request Detector*). Thanks to each one, pre-and post-treatment can be added to request. In this respect, we use these controllers in order to adapt application to context changes.

For adaptation management, we add three controllers to the container architecture. The first one, called *RPAD* (*ReactivProactivAdaptationDetector*), has to subscribe itself to the ContextAnalyser in order to be notified when relevant context changes occurs. The second one, called *ReactivAdaptor*, has to apply reactive adaptation when it is notified by the RPAD. The last one, called *ProactivAdaptor*, has to: (1) prepare a service request by adding adaptation request when a relevant context changes occurs, (2) change the invoked request by the prepared one, when a pre-request is intercepted, (3) adapt the invoked service by changing the source code using structural reflexion when a post-request is intercepted.

5 Related Work

Few research efforts in the domain of context-awareness have been addressed for creating context-aware middleware which offers a context model for context description. In this category, we can cite the RCSM project [8], the SOCAM project [9], and the ACAI project [10].

RCSM is an object-oriented middleware which enables the creation of context-aware applications. It provides an Interface Definition Language which generates the appropriate code for triggering application adaptation according to developer description. The CA-IDL language provides a limited context type description, without taking into account context interpretation nor proactive adaptation.

The SOCAM architecture aims to enable rapid prototyping of context-aware services in pervasive computing environment. It provides an ontology for context description. This middleware takes into account context acquisition and interpretation. It offers an API for context subscription but there is no mean to describe relevant context, so each service has to analyse and to filter the acquired context to detect relevant changes. Unlike CAMidO, the SOCAM middleware does not generate source code automatically from service designer description, it uses the inference engine provided by the ontology for adapting those services to context changes.

ACAI is a multi-agent infrastructure which allows context-aware application creation using an ontology model. ACAI enables interaction with sensors for context collection and interpretation. This latest action is performed using an inference module. ACAI does not provide any mechanism for relevant context nor adaptation action description. Adaptation is always carried out by the application level.

6 Conclusion

Context-awareness helps applications running on mobile devices to react into changing environment conditions. An efficient context representation and a context-aware middleware infrastructure are necessary for helping developers to create such type of applications.

In this paper, we described CAMidO, a context-aware middleware which provides an ontology meta-model for context description and application adaptation. The proposed meta-model allows context and relevant context description. All the described data are compiled using the *CAMidO compiler* for generating adaptation source code and rule files for relevant context detection. Relevant context is associated to an adaptation policy, the container has to apply this adaptation if a relevant context is detected.

The proposed middleware takes into account two types of adaptations, reactive and proactive adaptations. These adaptations are carried out by the component container to which new controllers were added.

Comparing to the existing context models solutions, CAMidO meta-model adds the possibility to describe actions of adaptation and sensors from which data are collected, relationships between different classes of the ontology are predefined by the proposed meta-model. These relationships allow the middleware to manage the collected context information and to adapt applications and services to context changes.

Currently we have implemented the CAMidO meta-model, the CAMidO Compiler, the *ContextInterpreter* and the *ContextAnalyser* components. The *CollectionManager* and the *ComponentAdapter* are still under development.

As a next step we intend to extend the proposed meta-model in order to infer adaptation actions which are not explicitly described by the application designer. For CAMidO middleware purpose, we intend to look further into the container architecture and the automatic distribution of applications context instances during the application deployment step.

References

1. OMG: CORBA Components Version 3.0 an Adopted Specification of the Object Management Group. OMG Document formal/02-06-65 (June 2002)
2. ObjectWeb: Open Source Software Community. Fractal project home page http://fractal.objectweb.org (2004)
3. McFadden, T., Henricksen, K., Indulska, J.: Automating context-aware application development. In: First International whorkshop on Advanced Context Modeling, Reasoning and Managenemt, UbiComp, Nottingham, England (September, 2004)
4. Bokun, I., Zielinski, K.: Active badges, the next generation. Linux J. **1998** (1998) pp.3
5. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., F.Patel-Schneider, P., Stein, L.A.: Owl web ontology language reference. available at:http://www.w3.org/TR/owl-ref (2004. W3C Recommendation)
6. Ayed, D., Belhanafi, N., Taconet, C., Bernard, G.: Deployment of component-based applications on top of a context-aware middleware. In: IASTED, International Conference on software Engineering, Innsbruk,Austria (February,2005)
7. Vadet, M., Merle, P.: Open Containers in component-based platforms. In: Proc. Day Emergent Components Topic, Besancon, France (2001) in French.
8. Yau, S.S., Karim, F., Wang, Y., Wang, B., Gupta, S.K.S.: Reconfigurable context-sensitive middleware for pervasive computing. IEEE Pervasive Computing **1** (2002) 33–40
9. Gu, T., Pung, H.K., Zhang, D.Q.: A middleware for building context-aware mobile services. In: Proceeding of IEEE Vehicular Technology Conference, Milan, Italy (2004)
10. M.Khedr, A.Karmouche: Acai: Agent-based context-aware infrastructure for spontaneous applications. Journal of Network and Computer Applications **Volume 28, Issue1** (2005) pages 19–44