Towards a Model Driven Development of Context-aware Systems for AmI Environments

Estefanía Serral, Pedro Valderas, Javier Muñoz, Vicente Pelechano

Departamento de Sistemas Informáticos y Computación
Technical University of Valencia
Camí de Vera s/n, E-46022, Spain
{eserral, pvalderas, jmunoz, pele}@dsic.upv.es

Abstract. In this work, we introduce a software engineering method for AmI applications which is based on a model driven strategy. This method allows us to describe an AmI application a high level of abstraction by means of a set of models and then automatically obtain code from these models by following an automatic code generation strategy. To do this, a method proposed by authors in previous works is extended in order to fully support AmI applications properties. The introduced extensions are: (1) a set of models that allow us to represent the context information at conceptual level and (2) a strategy to allow the system infers knowledge in execution time to anticipate user actions and adapt the system according to the context data. Finally, we want to mark that this method allows us to provide the development of AmI system with the benefits of using a software engineering method.

1 Introduction

Ambient Intelligence (AmI) is a new computing paradigm which tries to make reality the vision of Weiser [12]. Ambient Intelligence systems consist of mechanics, electronics, and software. The system intelligence, in particular, is realized via software. AmI systems are characterized by a constant evolution of hardware and software which force these software products (AmI applications) to be highly evolvable and adaptable in order to allow them to be executed in several kinds of devices with different resource capabilities. The use of engineering methods is crucial to properly developing this type of software.

However, current efforts in AmI are mainly focused on aspects such as the development of new implementation technologies, communication protocols or Artificial Intelligent algorithms. Few efforts have being made in order to create development methods that support the development of Ambient Intelligent application from a software engineering perspective. Thus, AmI applications are usually developed by ad-hoc solutions, which make their maintenance and further adaptation or evolution difficult.

In this work, we introduce a software engineering development method for AmI applications. This method is based on a model-driven strategy [11]. Our main objective is to provide developers with mechanisms that allow them to represent an AmI application in an abstract way (in a model) and then automatically generate (by a code generation tool) the corresponding code. This method provides the development of AmI applications with benefits that are historically well known in the software engineering community: high productivity, high quality, quick adaptability, easy maintenance.

In order to obtain all these benefits in the development of AmI applications, we extend the model-driven development method for pervasive systems that is presented in [9]. This method provides us with: (1) PervML, a domain-specific modelling language for pervasive systems and (2) a code generation strategy based on the Software Factories (SF) [10] and Model Driven Architecture (MDA) [11] philosophies which obtain Java code based on the OSGi Framework [9] from models. In order to fully support the characteristics of an AmI application, we extend this method with:

- A set of models that allow us to represent the context information at conceptual level. By context information, we mean information such as location information, user profiles, privacy and security policies, etc.
- A strategy that allows the system to infer knowledge from the context in run time. This allows the system to learn from people's behaviour in order to adapt itself to better support this behaviour (for instance, by anticipating the user actions). We support this inference with an OWL-based ontology (which represents concepts such as user, action, location, service, etc) and a set of rules that analyzes the descriptions defined by means of ontology concepts.

The rest of paper is organized as follows: Section 2 presents the development process proposed by our MDD method for AmI applications. Section 3 analyzes the concept of context information and gives an overview of PervML (the modelling language for pervasive systems). Section 4 introduces three new models which allow PervML to properly capture context information. Section 5 introduces the proposed strategy to infer knowledge from the user actions in run time. Section 6 introduces the related work about modelling methods and reasoning systems in the context of AmI. Finally, conclusions and further work are commented on in Section 7.

2 The Method for AmI Systems in a Nutshell

In [9] a method for the development of a software factory for building the core of AmI systems is proposed. This method applies the guidelines defined by: Model Driven Architecture (MDA), which is supported by the Object Management Group (OMG); and Software Factories, that is supported by Microsoft. Following these guidelines, the method provides (1) a modelling language (PervML) for specifying pervasive systems using conceptual primitives suitable for this domain, (2) an implementation framework that provides a common architecture for all the systems that are developed using the method, and (3) a transformation engine that translates PervML specifications into Java code.

In order to increase the expressiveness of the generated systems and to be able to develop context-awareness systems for AmI ecosystems, we have extended both PervML and the implementation strategy.

As Figure 1 shows, on the one hand, a set of models has been added to PervML that allows us to properly represent the context information (contribution 1). Therefore, the transformation engine that translates the PervML specifications into Java code of the context-awareness system for AmI can be applied. On the other hand, we have developed the PervML ontology, and have proposed a strategy to allow the system to infer knowledge from the context using the PervML ontology, in order to adapt the system to the needs of users (contribution 2). This strategy consists of automatically transforming

the PervML models in OWL specification according to PervML ontology. The Java code keeps this OWL specification updated according to run time information. Finally, an OWL reasoner, which is integrated with pervasive system, is used to infer knowledge about system data in order to adapt the system accordingly.

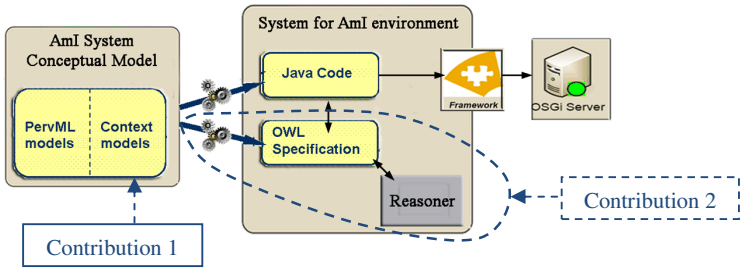

Fig. 1. Proposed method

3 Context Information and PervML

In this section we introduce both the concept of context and a brief overview of the main characteristics of the PervML models. Furthermore, we identify the context information that can be captured in these models and the context information that is not supported.

3.1 The concept of context

In this paper we consider that context is the state in which the system is, the relevant information that allows the system to react accordingly, making the system more productive and efficient. In order to identify this information, we have based on SOUPA ontology [2]. SOUPA is the most influential published ontology model for supporting knowledge sharing, context reasoning and interoperability in pervasive computing systems. It is frequently cited as a good example of pervasive computing ontologies too.

Therefore, according to this ontology, the context information is made up of: 1) information about system users: name, age, address, native language, etc.; 2) user preferences or attitudes, such as beliefs, desires or intentions; 3) space information; 4) services available to the user; 5) privacy and security policies that indicate what operations each user can execute; 6) temporal information: date and time, holiday, working day, etc.; 7) user mobility; 8) user actions: what the user is doing at present moment and what the user has done in the past. The actions performed in the past are necessary to predict the next action or to memorize and to reproduce scenes (repetitive sequences of actions within a certain time interval) at the opportune moment.

3.2 PervML

PervML [8] is a language designed to provide pervasive system developers with a set of conceptual primitives that allow them to describe the pervasive system independently of the technology. PervML promotes the separation of roles where developers can be categorized as system analysts and system architects.

On the one hand, systems analysts capture system requirements and describe the pervasive system at a high level of abstraction by using the service metaphor as the main conceptual primitive. The analysts construct three graphical models (the Services Model, the Structural Model and the Interaction Model), which constitute what we called the Analyst View. In these models, the analyst describes (1) the types of services that the system must support (by describing its interfaces, the relations among services and a state transition diagram which depicts the behaviour of each type of service); (2) the different components that provide the identified services of the system; and (3) how these components interact to each other.

On the other hand, System architects specify which devices and/or existing software systems support the system services. We refer to these elements (devices and software systems), as binding providers because they bind the pervasive system with its physical or logical environment. Therefore, the system architect constructs three other models (the Binding Providers Model, the Component Structural Specification, and the Components Functional Specification), which constitute the Architect View. In these models the system architect describes (1) the types of binding providers (by describing its interface), (2) the binding providers that are used in each component, and (3) the actions that must be carried out when each operation of each component is invoked.

Context in PervML. From the point of view of context information, PervML only provides support for specifying the available services and the locations of these services, although it does not allow us the different locations to be related. Thus, context information is not properly considered by PervML.

According to the concept of context introduced in Section 3.1, we can identify two type of context information: (1) Those that is available when the system is being developed (when we are creating the PervML models) such as information about users, the user preferences, locations where the system is going to be deployed, privacy and security policies, etc.; and (2) those that is available in run time such as temporal information, user mobility and actions that users perform. Thus, for the context to be properly modelled: (1) PervML must allow us to specify all context information available in modelling time and (2) the code generated from this specification must provide support for handle run time information. Next, we explain the extensions introduced to support this.

4 Including context-awareness in PervML

This section explains how context information is introduced in PervML. To do this, and according to SOUPA, the concepts of users, policies and space information must be added to the system specification, which will be specified by the system analyst. Thus, in the next subsections we present the User Profile Model to specify policies, the User Model to describe the system users, and the Location Model to describe the space information.

4.1 The User Profile Model

This model has been introduced to specify behaviour and privacy policies. It allows the system analyst to create a policy that can be applied to a user or a set of users.

With this model, the system analyst can specify the types of user, indicating the service operations that are available for each type or profile. To do this, we associate a list of perations to each type of user. The analyst can specify this list of operations in different ways: 1) by adding a service type so that every operation of every component that provide that service will be allowed; 2) by adding a component so that every operation of that component will be allowed; 3) by adding a service operation to allow this service operation to be permitted in every component that provide this service; or 4) by adding a component operation to be permitted this operation of this component is allowed.

Furthermore, inheritance relations can also be established in this model. These relations allow us to define capacities of a type of user taking the capacities of a defined type of user as a basis. We can do it in two ways: 1) By adding new capacities to capacities of the parent type of user. We use this way if the child type of user can execute more operations than the father type of user. The actions that the child can execute but father can not are denoted by "+". 2) By removing capacities of the parent type of user. We use this way if the child type of user has capacities more restrictive than the father type of user capacities. The actions that the father can execute but the child can not are denoted by "-".

For instance, in the example of Figure 2 the type *father* can execute operations related to the services Lighting, GradualLighting and BlindManagement. The type *child* can execute the same operation except the operations of the BlindManagement service.

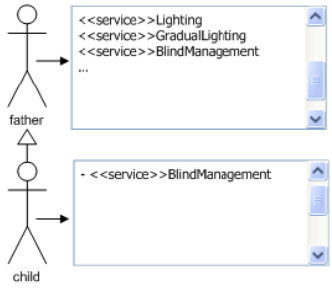

Fig. 2. User Profile Model

This model provides support for the privacy, the security and the views of the system, since users can see and execute only system actions that they are authorized to use.

4.2 User Model

This model has been introduced to specify user information. It contains the personal data of each one of these users: name, surname, sex, marital status, etc.; the contact information: email, movil, telephone, direction, etc.; and the social relations, i.e., information related to people that he user knows.

In this model the analyst must indicate the policy that is associated to the user, which has been previously specified in the User Profile Model.

4.3 Location Model

The Location Model describes the different areas where the user can move or where services can be located. It is specified by means of an UML package diagram. Each package represents a certain area, and the hierarchy between packages symbolizes the space hierarchy between the areas. Also, two types of associations can exist between the areas: adjacency and mobility (or accessibility). Adjacency means that the zones are one next to the other, whereas mobility is the possibility to go from one area to another. Adjacency is represented by a line between two areas. Since mobility implies adjacency, it is represented by adding arrows to the line between two areas.

Figure 3 shows an example of the Location Model. It models the locations of the first floor of a house that has four areas: Hall, LivingRoom, Kitchen, and Garden. The figure shows that, for instance, Hall and Garden are adjacent but a user can not go from one to another, however, Hall and LivingRoom are adjacent and user can go from one to another too. This model allows us to infer information such as transitive relations (for example, if the Kitchen is on Floor1 and Floor1 is in House1, we know that Kitchen is in House1).

In the component structural model, each component is related to its physical location.

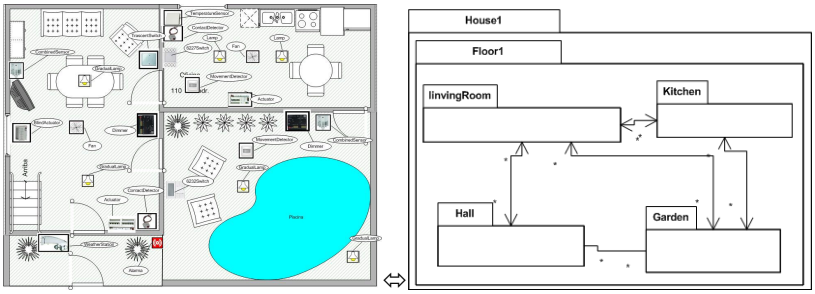

Fig. 3. Location Model of Floor1

5 Extracting Knowledge from AmI Systems in Run Time

In this section, we introduce a strategy to extract knowledge from the pervasive system in run time. This strategy allows the system to infer knowledge from the context data in order to anticipate user actions and adapt the system according to the context data. To do this, all the information about the system is stored according to the PervML ontology in OWL. OWL enables automated reasoning, and has the capability of supporting semantic interoperability to exchange and share context knowledge between different systems. It is also more expressive than other ontology languages and is an open W3C standard.

Thus, we present first, the PervML ontology that is defined in OWL. Next, we present the OWL reasoning strategy that allows the system to infer knowledge from the context data.

5.1 PervML ontology

The PervML ontology is detailed in this section through the description of the most important concepts, their properties and the relations among these concepts. We graphically show these concepts in Figure 4 by using the approach presented by Al-Muhammed et al [6]. According to this approach, two kinds of concepts can be defined: lexical concepts (enclosed in dashed rectangles), which represent the properties of each class; and nonlexical concepts (enclosed in solid rectangles), which represent the ontology classes. Figure 4 also shows a set of relationships among concepts, which are represented by connecting lines, such as Component has Trigger. The arrow connection represents a one-to-one relationship or a many-to-one relationship (the arrow indicates a cardinality of one) and the non-arrow connection represents a many-to-many relationship. The small circle near the source or the target of a connection represents an optional relationship.

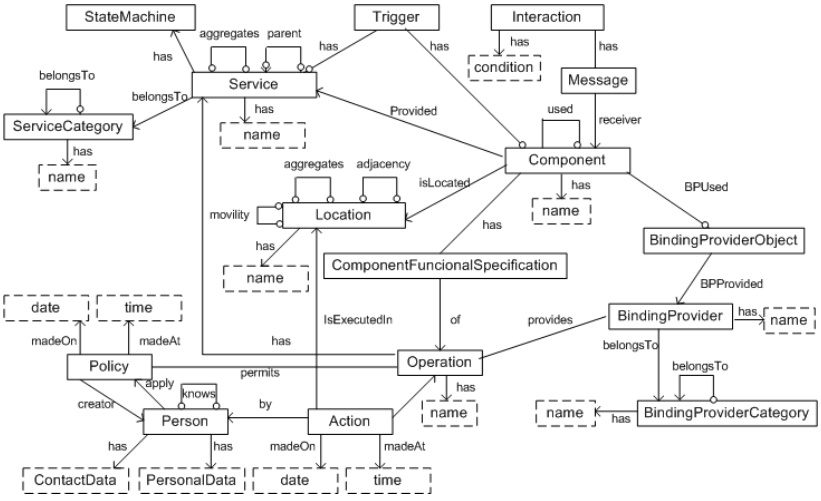

Fig. 4. A partial view of PervML Ontology

For instance, the concept Service, which is the entity that provides a coherent set of functionality which is defined in terms of atomic operations, is specified through its operations, its behaviour and its relations. As we can see in Figure 4, the properties that describe this primitive are: (1) Its name, which identifies it in the system. (2) The category to which the service belongs (illumination, multimedia…). Figure 4 indicates that its *belongsTo* relationship is a many-to-one relationship (i.e., each service only can belong to one category, but a category can group many services). (3) The set of operations provided by a service. (4) A set of triggers which allows the specification the proactive behaviour of the services. (5) The valid sequence of operations, which indicates the operations that can be invoked at a specific moment. (6) Its generalization service, if it exists. (7) And the set of services that the service aggregates, i.e., the set of services that the service needs to work. As Figure 4 shows this is a many-to-many relationship (i.e. a service can aggregate several services and a service can be aggregated by several services). The small circle near

the source and the target of the connection represents that it is not obligatory for a service to aggregate or be aggregated by other services.

As far as the rest of PervML concepts, see [8] for detailed information. Additionally, apart from PervML concepts, the PervML Ontology has, a set of classes based on the concepts introduced by SOUPA to represent the context data. This set of classes including Person, Policy, Location and Action classes. The Person class is defined to represent a set of all the users in the pervasive system. The Policy class is a set of rules that is specified by a user to restrict or guide the execution of actions. The properties of the Policy class are its author, its creation time, the set of people to whom the policy is applied, and an operation set that the policy permits. The Location class represents the different areas of the environment. Lastly, the Action class constitutes the execution of an operation. Its properties are: the operation that is carried out; the moment in which the action has been executed; and the person who executes it.

5.2 OWL Reasoning Strategy

In order to reason about the system and to allow its adaptation according to collected context data, our proposal is based on storing all system data in OWL according to PervML ontology. This data includes both type of information identified in section 3.2.

On the one hand, in order to store all the information available at modelling time in OWL we transform the PervML models in OWL automatically. To do this we base on the fact that the tool [13] which supports the creation of PervML models is developed using the Eclipse Modelling Framework (EMF) plugin [7]. EMF is a modelling set of tools and code generation facilities for specifying metamodels and managing model instances.

The PervML ontology has been developed using the EMF Ontology Definition Metamodel (EODM) plugin [7]. EODM is built on top of EMF and conforms to the ODM (Ontology Definition Metamodel) standard of OMG. EODM allows the user to create, modify, and navigate RDF/OWL models. Furthermore, EODM allow us to relate conceptual elements defined in the PervML models to the corresponding concept in the PervML ontology. For instance, we can indicate that each service define in the Service Model correspond to an individual form the concept service of the ontology. This allow us to obtain an OWL specification where appear classes (derived from the ontology concepts, e.g. Service) together with individuals (derived from the PervML models, e.g. Lighting).

As commented above, the OWL specification is automatically obtained. It is obtained from both the PervML models (defined using EMF) and the PervML ontology (defined using EODM) by means of an ATL transformation. The Atlas Transformation Language (ATL) has been used because it can be integrated into the eclipse framework and provides us with a set of model to model transformation tools.

On the other hand, the run time information is extracted by the own AmI system from their interaction with users. When the context information changes the system creates the corresponding OWL description of this change by using the Ontology concepts. For instance, when users perform an action the system creates an OWL individual of the concept action (see Figure 4). This individual will be moreover related with other OWL individuals such as date, time, etc. (individuals from concepts related with the concept action see Figure 4). Once the system has created this OWL description it is incorporated the OWL specification of the AmI system.

In this context, this specification in OWL, which is continually extended in run time, can be used by any reasoner or inference engine, such as RACER [14] (which is implemented in java), that infers additional and indirectly observable context information that can be used to reason about OWL classes, its properties and its relationships, and about ontology individuals. In this work, the reasoning about individuals is the most important because it allows us to deduce knowledge such as locations where a person can go (see Figure 5) or even to predict the following action of a user based on all executed previously actions by this user. The reasoner must be integrated into the AmI system and it can use rules that we can define by using, for instance, a declarative language like the one shown in the example in Figure 5 (Prolog). This type of rules allows the system to extract knowledge from the context, and automatically to adapt itself accordingly. Rule in Figure 5 tells the system where Peter can go from the location where Peter currently is.

```
Mobility(SittingRoom, (Kitchen, Garden, Hall))
IsLocated(Peter, SittingRoom)
Where_can_go (X, L) <- IsLocated(X,Y), Mobility (Y, L)
Peter can go to Kitchen, Garden or Hall
```

Fig. 5. Reasoning example

6 Related Work

This section presents an overview of some of the most important context modelling and reasoning systems that model context using ontologies. The architecture of the Context Managing Framework presented by Korpipää et al. [4] is comprised in four main functional entities: the context manager, the resource servers, the context recognition services, and the application. The ontology structure and vocabulary applied in the Context Managing Framework are described in RDF. CoBrA (Context Broker Architecture) [3] is an agent based architecture that supports context-aware computing in intelligent spaces. CoBrA has adopted an OWL-based ontology approach, and it offers a context inference engine that uses rule-based ontology reasoning. The SOCAM (Service-oriented Context-Aware Middleware) project introduced by Gu et al. [5] is another architecture for building context-aware mobile services. SOCAM divides a pervasive computing domain into several sub-domains and then define each sub-domain in OWL to reduce the complexity of context processing. SOCAM has also implemented a context reasoning engine that reasons over the knowledge base.

According to the study made in [1], ontologies are the most expressive models and best fulfil system requirements. But, none of these works attempts to model context and reason about it by applying the MDA and SF guidelines. Besides, none of them set reasoning out from the point of view of OWL individuals in order to infer relevant data from pervasive systems and adapt this behaviour accordingly. Our approach is more intuitive than traditional methods and allows us to focus on satisfying systems requirements. This is done by describing primitives of a high level of abstraction, which reduces development time because developers do not spend time solving technological problems.

7 Conclusions and Further work

In this work, we have presented a model driven development method for AmI applications which allows us to automatically generate context-awareness AmI systems from models. We have extended the model-driven development method for pervasive system proposed in [9] by introducing:

- A set of models that allow us to properly represent the context information at conceptual level. These models allow us to describe information related to location aspects and behaviour policies, as well as to the users that can interact with the system.
- A strategy to allow the system to infer context knowledge in real time. This allows the system to infer knowledge in order to adapt itself to the needs of system users (for instance, by anticipating the user actions). This strategy is based on OWL-based ontology which represents concepts such as user, action, location, service, etc., and a set of rules that inferring knowledge from context data.

As further work, we plan to 1) extend the expressiveness of PervML to allow modelling beliefs, desires and intentions; 2) specify and implement the rules for automatically transforming the new Perv-ML models in java code; 3) adapt automatically the system according to context data and 4) develop cases of study for context-aware systems for AmI environments.

References

1. Strang, T., Linnhoff-Popien C.: "A Context Modeling Survey". Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp 2004
2. Chen, H., Finin, T., and Joshi, A. (2004). "An ontology for context-aware pervasive computing environments". Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, 18(3):197–207.
3. Chen, H. (2004). "An Intelligent Broker Architecture for Pervasive Context-Aware Systems". PhD thesis, University of Maryland, Baltimore County.
4. Korpipää, P., Mäntyjärvi, J., Kela, J., Keränen, H., and Malm, E.-J. (2003). "Managing context information in mobile devices". IEEE Pervasive Computing.
5. Gu, T., Pung, H. K., and Zhang, D. Q. "A middleware for building context-aware mobile services". In Proceedings of IEEE Vehicular Technology Conference (VTC), 2004.
6. AL-Muhammed, M., Embley, D.W., and Liddle, S. "Conceptual Model Based Semantic Web Services". In ER 2005, volume 3716 of LNCS. Springer.
7. www.eclipse.org
8. Javier Muñoz, Vicente Pelechano, Joan Fons (2004). "Model Driven Development of Pervasive Systems", I International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES), Hamilton (Canada) pp. 3 - 14, ISBN: 952-12-1359-0
9. Javier Muñoz and Vicente Pelechano. "Building a Software Factory for Pervasive Systems Development". In CAiSE 2005 vol. 3520 of LNCS. Springer. pp 329–343, May 2005.
10. Jack Greenfield, Keith Short, Steve Cook, and Stuart Kent. "Software Factories". Wiley Publishing Inc., 2004.
11. Object Management Group. Model Driven Architecture Guide, 2003.
12. Weiser, M. (1991). The Computer for the 21st Century. Scientific American, 265(3):94–104.
13. Carlos Cetina, Estefanía Serral, Javier Muñoz, Vicente Pelechano. "Tool Support for Model Driven Development of Pervasive Systems". MOMPES 2007, Braga, (Portugal), pp. 33-41.
14. http://www.sts.tu-harburg.de/~r.f.moeller/racer/