Building a Software Factory for Pervasive Systems Development*

Javier Muñoz and Vicente Pelechano

Departamento de Sistemas Informáticos y Computación,
Universidad Politécnica de Valencia,
Camí de Vera s/n, E-46022, Spain
{jmunoz, pele}@dsic.upv.es

Abstract. The rise of the number and complexity of pervasive systems is a fact. Pervasive systems developers need advanced development methods in order to build better systems in an easy way. Software Factories and the Model Driven Architecture (MDA) are two important trends in the software engineering field. This paper applies the guidelines and strategies described by these proposals in order to build a methodological approach for pervasive systems development. Software Factories are based on the definition of software families supported by frameworks. Individual systems requirements are specified by means of domain specific languages. Following this strategy, our approach defines a framework and a domain specific language for pervasive systems. We use the MDA guidelines to support the development of our domain specific language and the automatic generation of the specific source code of a particular system. The approach presented in this paper raises the abstraction level in the development of pervasive systems and provides high reusable assets to reduce the effort in the development projects.

1 Introduction

Computing based systems growth is arriving to all environments of our daily life. Pervasive systems live around us providing services to the inhabitants of a home, the workers of an office or the drivers in a car park. We know that requirements for current and future pervasive systems involve a great diversity of types of services [14]. Such different services as multimedia, communication or automation services need hardware devices that different manufacturers provide. These devices live in several networks running on different platforms. The development of such systems is a very hard task because it should achieve devices interoperability in an heterogeneous environment in order to satisfy system requirements.

Therefore, there is a need of new solid engineering methods for developing robust pervasive systems. Recently, two compatible approaches have been proposed

* This work has been developed with the support of MEC under the project DESTINO TIN2004-03534 and cofinanced by FEDER.

O. Pastor and J. Falcão e Cunha (Eds.): CAiSE 2005, LNCS 3520, pp. 342–356, 2005.

for developing software systems in a highly productive and cost-effective way. Software Factories [4] and the Model Driven Architecture (MDA) [10] provide strategies for raising the abstraction level in the software development process and making affordable the development of complex systems. The application of the guidelines defined in this approaches to pervasive systems development can help to build better systems in an easier way than applying traditional methods. Software Factories focus on producing reusable assets that reduce the overall development time. On the other hand, MDA promotes the use of high abstraction level models which provide the system developers with an intuitive way for describing the system. These models should be automatically transformed to the final implementation code.

The work presented in this paper proposes a methodological approach to pervasive systems development following Software Factories principles and MDA guidelines. The contribution of is this work is double. On the one hand, we apply the Software Factories proposal to a concrete domain. Then, this work provides an application case that can be used as a recipe for the construction of Software Factories for other domains. On the other hand, our approach contributes to the state of the art in pervasive systems development. We provide a model driven development method for the specification and implementation of pervasive systems. Our approach establishes a methodological framework for automating the construction of high-quality pervasive systems in a productive way

The structure of the paper is the following: Section 2 briefly introduces the Software Factories and MDA approaches. We justify the application of these proposals to Pervasive Systems development. Section 3 describes Pervasive Systems main characteristics and it presents our point of view for developing these kind of systems. We introduce a Pervasive System for a meetings room in order to illustrate our approach. Section 4 describes our application of Software Factories and MDA to Pervasive Systems. We present our strategy to apply MDA guidelines into our methodological approach. We propose techniques for every MDA building block. The Pervasive Modeling Language (Perv-ML), a modelling language for describing Pervasive System using high abstraction level constructs, is introduced. Next, a framework for developing Pervasive Systems is presented. Finally, section 5 includes some conclusions and further work.

2 Software Factories and MDA

A **Software Factory**, as defined in [4], is *a software product line that configures extensible tools, processes and content [...] to automate the development and maintenance of variants of an archetypical product by adapting, assembling and configuring framework-based components.* Therefore, Software Factories focus on the development of similar systems encouraging the reuse of architectures, components and know-how.

On the other hand, MDA is, as described in the IEEE Software special issue on Model Driven Development [8], *"a set of OMG standards that enables the specification of models and their transformation into other models and complete*

systems.". Following this approach, system developers build high level abstraction models (called Platform Independent Models, PIM) and transform them obtaining models that directly represent the final software product (called Platform Specific Model, PSM).

Therefore, there is a natural integration of these two approaches. MDA techniques can be used to support the development of domain specific languages for building high level abstraction models. Then, these models can be transformed in order to obtain the specific source code of a system in the context of a family of systems.

Although the use of OMG languages is explicitly avoided and criticized in [4], we think that the use of standards is a key aspect for the success of these approaches, even when the standards are not as good as we would like.

In short, we are interested in the **strengths of both approaches**:

- *from Software Factories* we get their focus on reuse by means of domain specific development.
- *from MDA* we get their focus on high level abstraction models and automatic code generation.

In order to describe this specific domain, next section points out pervasive systems main characteristics.

3 Pervasive Systems

Pervasive systems try to build environments where computation elements disappear from the user point of view but their functionality is still provided. This vision was initially described by Weiser [15] in the early 90s and it is based on the construction of computing-saturated environments properly integrated with human users. The big challenge of this vision is the integration of several existing technologies (handheld computers, broadband communications, sensor devices, etc.) in an homogeneous whole. The development of such a kind of systems requires the contribution of several engineering and research fields: hardware designers, human-computer interaction experts, software engineers, etc. We can find pervasive systems in environments like cars, offices, public building and, of course, our homes.

Requirements for current and future pervasive systems involve a great diversity of types of services [14]. Such different services as multimedia, communication or automation services need hardware devices that different manufacturers provide and external software systems. These elements live in several networks running on different technological platforms, but they can not satisfy isolatedly all system requirements. The elements that compose the system must work together for achieving some system goals. Therefore we can distinguish two sources of service providers: **commercial off-the-shelf (COTS) elements**[1] and the **software system** that integrates all the elements of the pervasive system.

[1] We extend the definition of COTS to include hardware devices

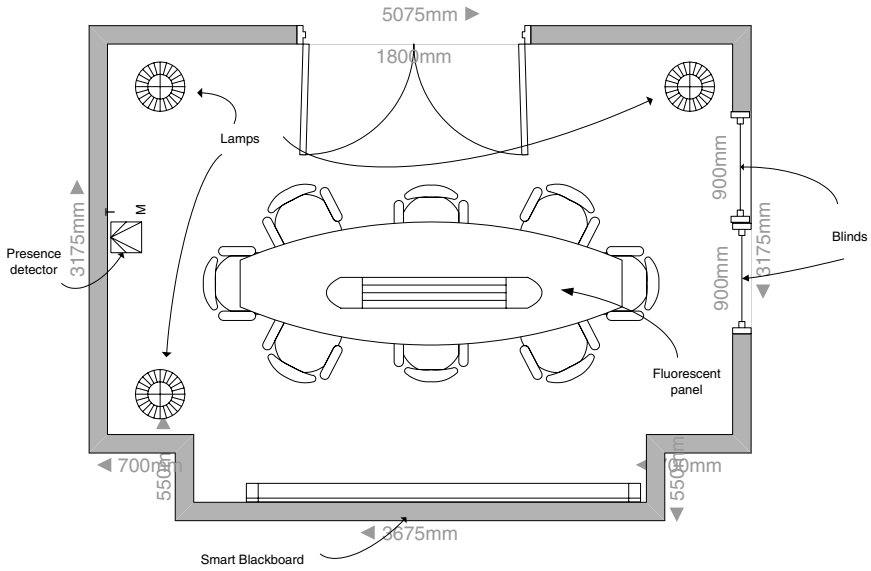

Fig. 1. The map of a smart meetings room

Considering this point of view, the development of a pervasive system consists of :

- **The selection of the suitable COTS devices or external software systems.** These elements should provide the services that users require either isolatedly or interacting with other elements.
- **The development of the software system that integrates the external elements in order to provide the services that users require.** The development of that software may imply the use of different technologies but some gateway technology should exist.

We describe a pervasive system for a meetings room in order to illustrate our approach. In such a system, depicted in Fig. 1, users require services like *lighting management* by *rooms presence*, *blinds management* or *drawings sharing*. Users do not mind what devices compose the system, they just need a specific functionality. System architects deal with selecting the most suitable devices (like *lighting bulbs* or a *smart board* in our case of study) for providing that functionality.

4 A Software Factory for Pervasive Systems Development

As outlined in section 3, the development of a pervasive system implies the use of many different technologies in order to satisfy all users requirements [13, 6]. Usually these technologies provide low abstraction level constructs to the developer.

Therefore, applying a MDA approach to pervasive systems supposes jumping a very wide abstraction gap that must deal with the technology heterogeneity.

Following the Software Factories approach, a framework for pervasive systems should be developed applying domain engineering principles. This framework raises the abstraction level of the target platform and, therefore, the amount of code is sensibly reduced.

Thus our proposed methodological approach to pervasive systems development is based on:

- the construction of a domain specific language for the description of pervasive systems.
- the construction of a framework that raises the abstraction level by providing similar constructs to those defined by the domain specific language.
- the definition of mappings or rules for the transformation of models, that are built using the domain specific language, to code that fulfils the defined framework.

Next subsections describe both the MDA point of view of the proposed approach and the main architecture of the framework for pervasive systems. Subsection 4.1 presents the techniques for the construction of the domain specific language and the definition of the mapping rules. Subsection 4.2 introduces the implementation framework for developing pervasive systems.

4.1 MDA for Pervasive Systems

As we have justified in section 2, the MDA approach can be used in a Software Factory to support the development of domain specific languages and the automatic code generation step. The standard defines several building blocks for the definition of MDA based methods, but it does not specify concrete techniques to be used in each step. In order to put MDA in practice we should provide concrete techniques for each building block. These techniques must be defined using OMG standards. Our approach proposes the following techniques for applying MDA (see Fig. 2) to pervasive systems development:

1. A **precise language for building Platform Independent Models** (PIMs). This is the domain specific language for precisely describing pervasive systems using high abstraction level constructs. We have defined the Pervasive Modeling Language (Perv-ML) (outlined next) in order to build PIMs.

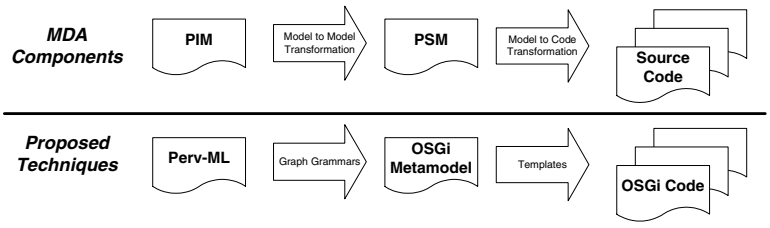

Fig. 2. MDA building blocks and our proposed techniques for pervasive computing

2. One or many **modelling languages for building Platform Specific Models** (PSMs). The conceptual primitives of these languages must be direct representations of constructs of the technology they model. In our case, the target platform is the framework for pervasive systems developed as result of the domain engineering activity. As we will see next, this framework is tightly based on OSGi [7]. OSGi is a standard defined by the Open Service Gateway Initiative (OSGi) that describes a framework that was initially created for hosting software of residential gateways. Then, we have defined an OSGi metamodel for building Platform Specific Models.
3. **PIM to PSM transformations**. These transformations define how a PIM can be converted to a PSM. Currently, model transformations is a hot research topic. We apply graph grammars for defining the transformations from Perv-ML to OSGi.
4. **PSM to source code transformations**. Finally, the code generation from the PSMs is the last step of the development method. We are applying templates to the elements of models in order to obtain the source code.

4.1.1 Pervasive Modelling Language (Perv-ML): The PIM Language

Perv-ML is a language designed with the aim of providing the system analyst with a set of constructs that allow to precisely describe the pervasive system. Perv-ML promotes the separation of roles where developers can be categorized as analysts and architects. Fig. 3 shows the language organization. The dashed arrow of Fig. 3 defines the construction order of the conceptual models that our approach proposes. In short, systems analysts capture system requirements and describe the pervasive system at a high level of abstraction using the service metaphor as the main conceptual primitive. Analysts build three graphical models that constitute what we call the **Analyst View**. On the other hand, system architects specify what COTS devices and/or existing software systems realize system services. Architects build other three models that constitute what we call the **Architect View**. Next we give a more detailed description of the language.

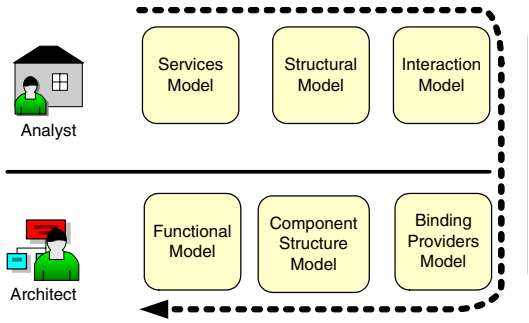

Fig. 3. The six models of Perv-ML

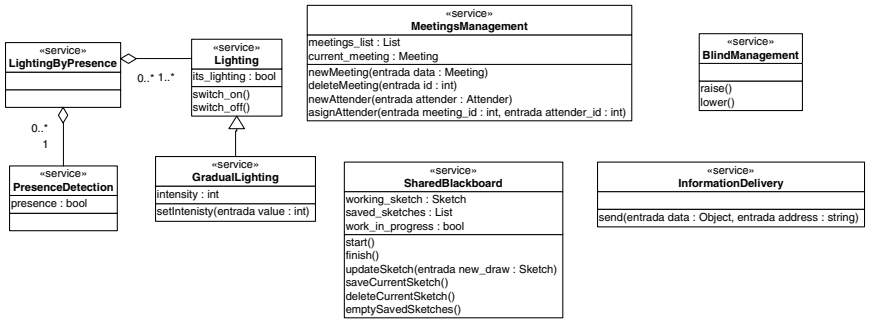

Fig. 4. Meetings room Services Model

The Analyst View. The Analyst describes a pervasive system specifying a set of functional elements that provide a specific set of services that the user of the system requires. Those functional elements are what we call service instances. For instance, if the meeting room described above has two binds and any user wants to control them independently, the pervasive system must provide two elements (instances) that provide the bind management service. Following this approach we propose a step previous to the building of the Pervasive System Conceptual Structure. In this first step, we introduce the **Services Model** where the analyst defines services and their relationships. Perv-ML uses and extends UML Class Diagram for representing the description of the services, and the State Transition Diagram for modelling the behaviour. Fig. 4 shows the Service Model of our meeting room.

Analyst defines the pervasive system functional structure in the **Structural Model**. This model specifies the service instances of the system which are represented by a component. Perv-ML provides components as abstractions of the low-level elements that realize the services. Every system component provides one of the services described in the Services Model. In Fig. 5. we can see that the `LightingManagement` component has dependency relationships with the `MainLighting` and the `Presence` components due to the aggregation relation-

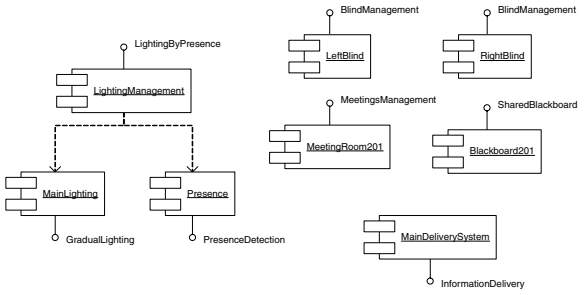

Fig. 5. Components that provide the services of our case of study system

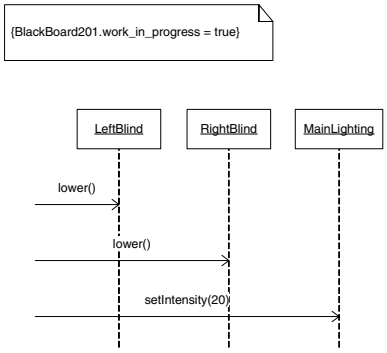

Fig. 6. An interaction that lowers blinds and sets lighting to 20% of its maximum intensity

ship defined in the Services Model. Perv-ML represents the Structural Model as a UML Component Diagram.

As we have said in section 3, system services must cooperate in order to satisfy all the system requirements. Analyst describes services cooperation in the **Interaction Model**. An interaction is a communication between services for providing a specific functionality, so analyst must describe as many interactions as joint functionality the system provides. Every interaction is described by an adapted UML Sequence Diagram, therefore the Interaction Model is composed by several sequence diagrams. Fig. 6 shows an interaction for suiting lighting when the blackboard service is being used. It lowers both blinds and it sets the lighting service at a 20% of its maximum power. This interaction takes place when somebody starts using the blackboard.

The Architect View. We need to build a detailed specification of the lower level artefacts that realize system services in order to have a complete and operative pervasive system description. We use the term *Binding Provider* for referring artefacts that the pervasive system manages to interact with its physical or logical environment. A *device*, a *sensor*, an *actuator* or an *external software system* can be binding providers. Architect describes every binding provider type that is introduced to implement system services in the **Binding Providers Model**. A type of binding provider represents a set of devices or software systems that provide a similar functionality without detailing manufacturer specific information. The Binding Provider Model is depicted using a stereotyped UML Class Diagram. Fig. 7 shows some binding providers of our meeting room. The usage of `Lamp` and `FluorescentPanel` actuators is different although both can be used for lighting a room.

The System architect uses the **Component Structure Specification** to specify the bindings providers that realize a component of the Structural Model. For instance, a component that provides a lighting management service can be realized by three lamps and a fluorescent panel. In such a case, the Binding Providers Model must contain the lamp description. See Fig. 8 of the Structure

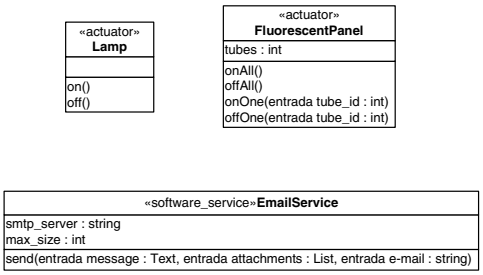

Fig. 7. Some elements of a Binding Providers Model

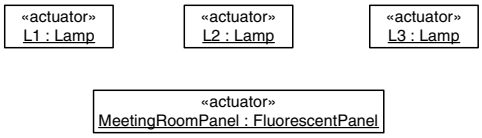

Fig. 8. Structure Specification of the `MainLighting` component

Specification for the `MainLighting` component included in our meeting room Structural Model (see Fig. 5).

Finally, architect must specify how every component operation is realized. In the **Component Functional Specification** architect defines the sequence of actions that the component realize when an operation is invoked. Architect specifies actions using the UML Action Semantic Language (ASL). ASL does not have an official concrete syntax, but many proposed syntaxes are available like the one by Kennedy Carter [16].

Using the Perv-ML approach the system is completely described in a technology and manufacturer independent way. When a new technology emerges, system description does not need to be modified. Moreover, if we want to use a device of a new manufacturer we only have to develop a driver that adapts its interface to the generic interface used in the Binding Providers Model. Even if the system architect decides to change a component specification, analyst view remains unmodified. We have isolated changes by means of stratification through abstraction levels.

4.1.2 OSGi Metamodel: The PSM Language

As described above, there are a lot of implementation technologies for developing pervasive systems. Using only a low-level technology for control (LonWorks, EIB, UPnP) , data (Ethernet, Bluetooth, WiFi) or multimedia (IEEE1394, HAVi) networks is not possible because of the diversity of services required, therefore we have selected OSGi, a middleware platform that has bridges to many of them and provides high-level constructs for building pervasive systems. This

middleware help us notably for filling the abstraction gap between the domain specific language and the target implementation technology.

The Open Service Gateway Initiative (OSGi) [7] is an association of companies, that includes Sun Microsystems, IBM, Oracle and Nokia, created with the aim of developing an open standard for service gateways. A service gateway is the platform where resides the software for providing home services. It manages home devices and it communicates with external networks. The standard defines Java APIs for libraries that the OSGi platform provides and several standard services like Logging, HTTP Server, Device Management, etc. Our own framework is built on top of this middleware using their runtime environment and services.

In order to integrate OSGi in the MDA phase of our development method, we have to create models which are built using OSGi concepts. We have developed an OSGi metamodel for specifying these concepts and their relationships. The models built with a OSGi-only metamodel cannot be seen as final implementation models because every OSGi concept is actually implemented as a Java entity. For instance, an OSGi Bundle is implemented as a JAR package, and an

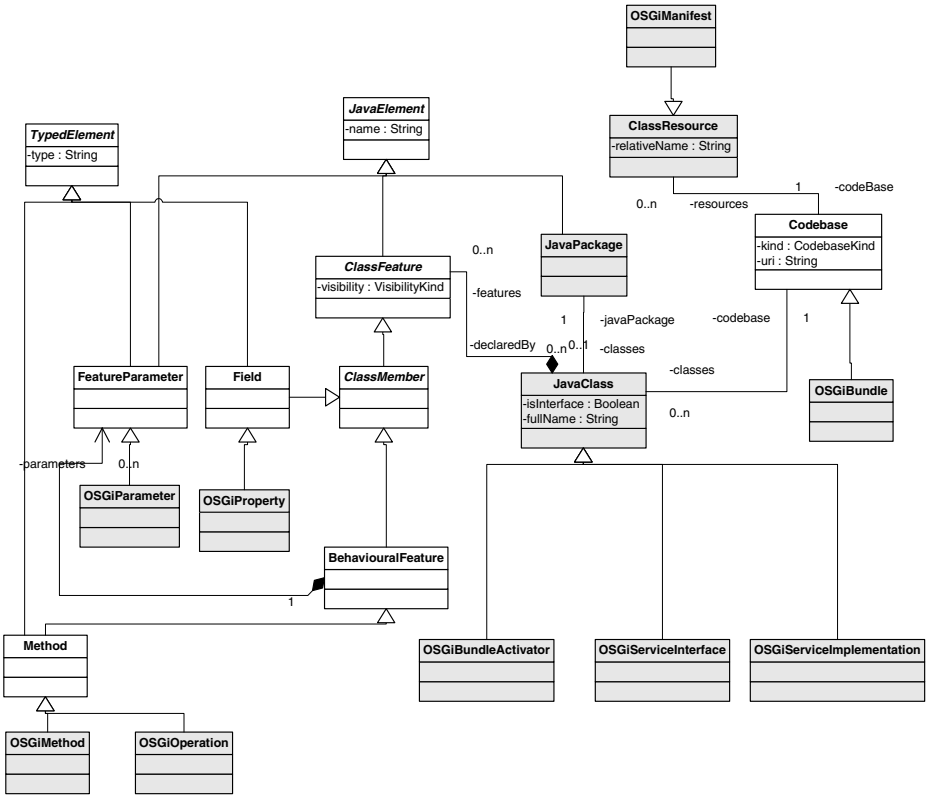

Fig. 9. An OSGi/Java metamodel

OSGi Service Implementation is implemented through a Java Class. Therefore, a fully functional modelling language for specifying OSGi based systems should include a complete Java metamodel. Then, the inclusion of OSGi concepts can be done as Java entities extensions with new specific constraints.

Our complete OSGi/Java metamodel is based on the Java metamodel developed by the NetBeans Community [2]. This metamodel has been adapted and extended to fit it in our needs. Fig. 9 shows a view of the Java metamodel with our extensions. Elements that are mapped from the OSGi metamodel have been depicted in grey.

4.1.3 Graph Grammars. Defining the Model Transformation Engine

As noted earlier, the definition of transformations between PIM and PSM involve jumping a wide gap between abstraction levels. Currently standards for the definition of transformations do not exist [2]. OMG published a *Request For Proposal* [9] in order to achieve a language for defining transformation between metamodels built with its Meta Object Facility (MOF). In the meantime, we are using graph grammars [3] as the model transformation engine. There exist many works [1, 5, 12] that propose graph grammars as a suitable technique for model transformation. From a mathematical point of view, a model can be seen as a graph where model elements are labelled nodes and the relationships between model elements are edges. In this way we can apply all the existing knowledge for defining graph transformations in order to achieve model transformations in the MDA context. Graph grammars have many advantages over other proposed techniques: a formal mathematical sound, algorithms for their application and a graphical representation for intuitively defining transformations.

Fig. 10 shows two rules for model transformation from Perv-ML models to OSGi-based models. Every rule is composed by a Left Hand Side (LHS), that defines a pattern to be matched in the source graph, and a Right Hand Side that defines the replacement for the matched subgraph. For instance, first rule says that when a Perv-ML `Component` element is found it must be transformed into a `Bundle` element and references to a Java `Class` and `Manifest` elements have to be created and linked to the `Bundle`. Following this approach, transformation from Perv-ML models to OSGi-based models is defined following a set of rules like those defined in this section.

4.2 A Framework for Pervasive Systems Development

The framework for pervasive systems has been developed for supporting Perv-ML, the domain specific language for this kind of systems. Therefore, the scope and approach of the framework is inherited from the language. This means that, as in Perv-ML, the framework is based on the assumption that the software of a pervasive system must integrate many devices and external software systems in order to provide the services that the users require. Following this approach, users should deal with services and the software is in charge of the management of

[2] http://java.netbeans.org/models/java/java-model.html

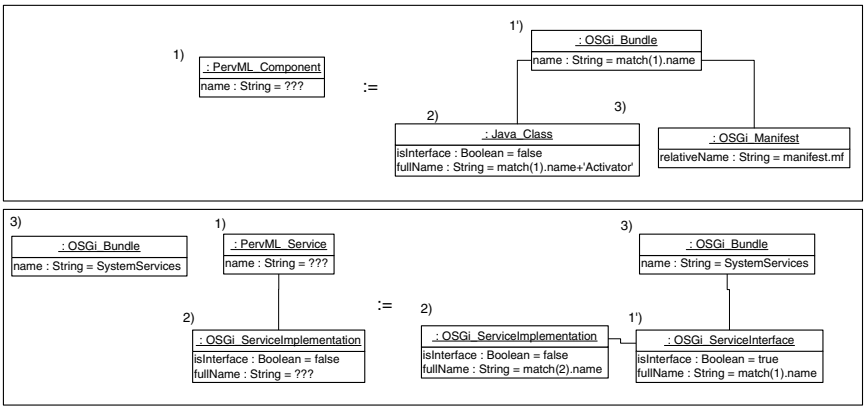

Fig. 10. Two rules that define models transformation

the devices or external systems for providing that services. Then, our framework provides implementation primitives for directly supporting Perv-ML conceptual primitives.

As described in subsection 4.1, we are using OSGi as platform for the development of pervasive systems. This technology fits smoothly in our approach and minimizes the abstraction gap to be filled. Many Perv-ML conceptual primitives maps directly to OSGi implementation concepts, so OSGi can be considered a key component of our framework.

Several architectures can be used when developing with OSGi. Fig. 11 shows the global structure of the systems developed with our method. Packages in the figure represent sets of resources (classes, interfaces, icons, etc.) with a common goal. Dashed arrows represent dependence relationships. For instance, the *Service 2* package requires some resource located in the *Device 1* package. We use a three-tier architecture adapted to this kind of systems. Layers of the architecture are described next.

4.2.1 User Interface Layer

The user interface layer is currently implemented as web pages using the HTTP Service integrated in the OSGi platform. We divide this layer in two components.

- The **main user interface** is the entry door to the system and is in charge of the organization of the access to the system services (by localization, by kind of service, by more used services, etc.) and security issues.
- The **individual services interface** manage the interaction of every particular service in the system. Services of the same type have the same user interface. For instance, in Fig. 11 services 2 and 3 (maybe lighting services) are managed by the user interface 2 (UI2).

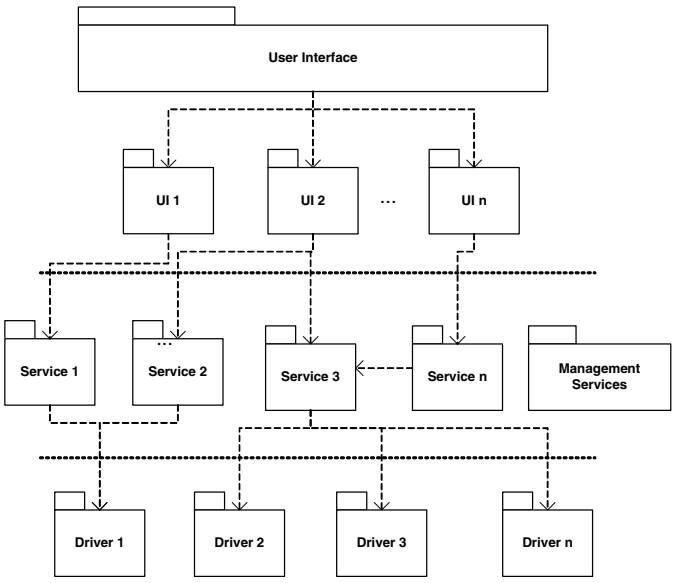

Fig. 11. Global architecture of the framework

4.2.2 Logical Layer

Elements in the logical layer can be classified in two groups attending their purpose:

- **Services for supporting the functionality specified in the Perv-ML model**. These services are implemented as Java Classes and registered as OSGi Services. All of them must implement the *PervMLService* interface for ensuring a proper execution of the system. This interface has operations for checking invariant constraints, managing concurrent execution, error handling, etc.
- **Services for the management of the system execution**. This set of services are in charge of ensuring global constraints satisfaction, checking trigger conditions, providing web services and other auxiliar functionality. These services are common to all the applications based on this framework.

4.2.3 Communication Layer

Finally, the **Communication Layer** manages the interaction of the pervasive system with its physical or logical environment. This layer is composed by drivers that are used by the services in the upper layer. Every device or external software system is repesented in this layer by a driver. Drivers in this layer are implemented as Java Classes and registered as OSGi Services.

For avoiding interface heterogeneity, a unified interface is used for all similar devices (or software systems). This means that, for instance, in our example

there is a unique interface for all lamps, all instant messaging systems or all video projectors. Then, the driver is in charge of adapting that interface to the actual device interface. The selection of the specific drivers for every generic device interface happens in model compilation time.

5 Conclusions

In this paper we have presented a methodological approach for the development of pervasive systems. Following the Software Factories strategy, our approach is based on the construction of a framework for a family of similar systems and a domain specific language (Perv-ML) for the specification of family members requirements. We follow the MDA standard for the definition of the domain specific language and the automatic code generation step. This merged approach can help to build better pervasive systems in an easier way than applying traditional methods.

We have experimented many of these benefits in the development of Information Systems. Our research group have developed a model driven method (called OO-Method [11]) with full code generation capabilities that has been implemented in the OlivaNova Model Execution System [3]. Our aim is to apply these successful ideas to pervasive systems development. This work was initially developed in the context of a R&D project together with Telefonica I+D. Perv-ML has been applied for the specification of applications for Smart Homes.

We are currently working on providing tool support for several steps of the method, like the construction of models using Perv-ML and the automatic transformations of that models. Another important ongoing work is the specification of the transformation rules from Perv-ML to OSGi code to implement the specific part of the framework. Finally, we are implementing pervasive systems with our framework in order to obtain feedback for tuning our proposal.

References

1. György Csertán, Gábor Huszerl, István Majzik, Zsigmond Pap, András Pataricza, and Dániel Varró. VIATRA: Visual automated transformations for formal verification and validation of UML models. In Julian Richardson, Wolfgang Emmerich, and Dave Wile, editors, *Proc. ASE 2002: 17th IEEE International Conference on Automated Software Engineering*, pages 267–270, Edinburgh, UK, September 23–27 2002. IEEE Press.
2. Krzysztof Czarnecki and Simon Helsen. Classification of model transformation approaches. In *2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, 2003.
3. H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook on Graph Grammars and Computing by Graph Transformation*, volume 2 Applications, Languages and Tools. World Scientific Publishing Co., Inc., 1999.

[3] http://www.care-t.com/

4. Jack Greenfield, Keith Short, Steve Cook, and Stuart Kent. *Software Factories*. Wiley Publising Inc., 2004.
5. Reiko Heckel, Jochen Küster, and Gabriele Taentzer. Towards automatic translation of UML models into semantic domains. In *Proc. AGT 2002: Workshop on Applied Graph Transformation*, pages 174–188, 2002.
6. M. Hwang, Y. Jeon, and J. Kim. Standarization activities and technology competitors for the in-home networking. In *Internation Conference on Communication Technology*, 1998.
7. Dave Marples and Peter Kriens. The Open Services Gateway Initiative: An Introductory Overview. *IEEE Communications Magazine*, 39(12):110–114, 2001.
8. Stephen J. Mellor, Anthony N. Clark, and Takao Futagami. Guest editors' introduction: Model-driven development. *IEEE Software*, 20(5):14–18, 2003.
9. Object Management Group. OMG MOF 2.0 Query, Views, Transformations Request for Proposals.
10. Object Management Group. Model Driven Architecture Guide, 2003.
11. Oscar Pastor, Jaime Gómez, Emilio Insfrán, and Vicente Pelechano. The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. *Information Systems*, 26(7):507–534, 2001.
12. Shane Sendall. Combining Generative and Graph Transformation Techniques for Model Transformation: An Effective Alliance? In *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*, 2003.
13. Kenneth Wacks. The successes and failures of standardization in home systems. In *2nd IEEE Conference on Standardization and Innovation in Information Technology*, 2001.
14. Roy Want, Trevor Pering, Gaetano Borriello, and Keith I. Farkas. Disapearing Hardware. *Pervasive Computing*, 1(1), 2002.
15. Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, Sept. 1991.
16. Ian Wilkie, Adrian King, Mike Clarke, Chas Weaver, and Chris Rastrick. *UML ASL Reference Guide*. Kennedy Carter Limited, 2001.