Tool Support for Model Driven Development of Pervasive Systems*

Carlos Cetina, Estefanía Serral, Javier Muñoz, Vicente Pelechano
Department of Information System and Computation.
46022 Cami de Vera s/n.
Technical University of Valencia, Spain
{ccetina,eserral,jmunoz,pele}@dsic.upv.es

Abstract

This work presents the PervML Generative Tool (PervGT) that supports a model driven method for the development of pervasive services in ubiquitous environments. The tool, which is based on the Eclipse platform, provides facilities for the graphical description of pervasive systems using PervML, a UML-like modeling language. Once the pervasive system is specified, the PervML model is used as input to a transformation engine that generates source code and other implementation assets. This generated code extends an OSGi-based framework in order to build the final pervasive applications.

1. Introduction

As stated in [8], "*the next few years will determine the success or failure of Ubiquitous Computing research"*. Pervasive systems are moving from the academy to the industry, and this evolution implies that the systems under development are more complex and with more quality requirements than in research prototypes. Thus, solid engineering methods are needed in order to produce robust systems in an efficient way. If we fail in providing specific methods which solve the challenges that pervasive systems have introduced, it will be very hard to completely achieve the vision that was disseminated by Weiser.

In order to deal with the increasing complexity, this kind of methods must also increase the abstraction level of the concepts that are used for building pervasive systems. Following this strategy, the developers describe the system using primitives which are suitable for the problem domain (specifying what the system must do) instead of using technological concepts (specifying how the functionality must be implemented). In this context, a model driven approach seems a good option, since this strategy is supposed to provide this characteristic.

A widely accepted statement is that tools are needed in order to put in practice in an effective way all the techniques involved in the construction of a Model Driven Software Development (MDSD)-based method. Without tools which automate the steps that must be carried out during the application of such kind of methods, most of the promised benefits can not be obtained.

This work introduces a tool which supports a model driven method for the development of pervasive services in ubiquitous environments [3]. The method is based on the specification of the pervasive system using PervML [2, 5], a high level of abstraction UML-like language. The tool, which is deployed as an Eclipse plug-in, provides facilities for the graphical description of pervasive systems. Then, the PervML specifications are automatically translated into Java code by a transformation engine that is integrated in the development environment. The generated code extends an OSGi-based framework in order to build the final pervasive applications.

The paper is organized as follows: first Section 2 briefly presents the model driven method which is the basis of the tool. In Section 3 the technologies that have been used for implementing the different components of the tool are presented. Section 4 introduces the PervML Generative Tool. Subsection 4.1 presents the tool architecture and describes every tool building block, whereas Subsection 4.2 shows the tool from the user point of view. Finally, Section 5 contains the conclusions, related works and future lines of research.

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

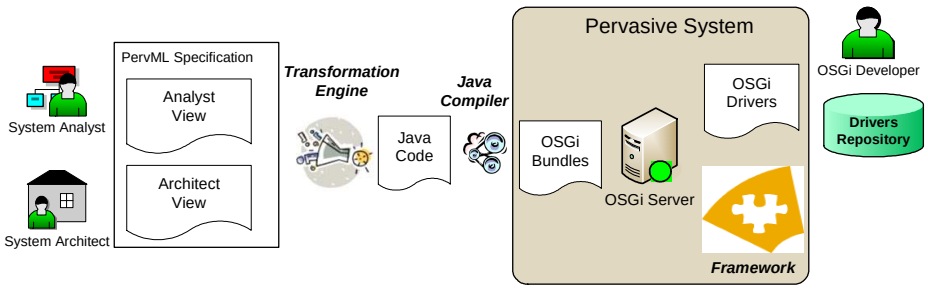

Figure 1: Model driven method for the development of pervasive services that is supported by the tool

2. A Model Driven Development Method for Pervasive Systems

The proposed method for the development of pervasive systems applies the guidelines defined by the Model Driven Architecture (MDA), which is supported by the Object Management Group (OMG), and the Software Factories, which is supported by Microsft. Figure 1 presents an overview of the model driven method for the development of pervasive services in ubiquitous environment [3] that is supported by the tool. The figure shows the developers which are in charge of performing the method steps and the assets that are produced during the development process.

The method provides (1) a modeling language (PervML) for specifying pervasive systems using conceptual primitives suitable for this domain, (2) an implementation framework which provides a common architecture for all the systems which are developed using the method, and (3) a transformation engine that translates the PervML specifications into Java code.

The implementation framework, which is introduced in [4], provides similar abstract classes to the PervML conceptual primitives (Service, Trigger, Interaction, etc.) in order to facilitate the translation process. The framework has been build on top of the OSGi middleware [1], which is a standard Java-based and dynamic execution environment for services. This technology has bridges to many of the technologies that can be used in pervasive systems and provides high level of abstraction implementation constructs.

In summary, a development team must carry out the following steps in order to produce a pervasive system using our method:

1. The system analyst specifies the system requirements using the service conceptual primitive. The system analyst uses three kinds of PervML models in order to describe (1) the kind of services available on the system, (2) the number of services which are available in every location and (3) how they interact when some condition holds.
2. The system architect selects the kind and number of devices or software systems that are more suitable in order to provide the services specified by the analyst. The selection could have into account economical reasons or constraints in the system physical environment. The pervasive system architect uses other three PervML models for describing (1) the kind of devices or software systems that are used for providing the system services, (2) the specific elements that are going to implement every service and (3) the actions that the device or software systems must carry out for providing every service operation.
3. An OSGi developer implements the drivers for managing the devices or software systems which were selected by the system architect. These drivers provide access from the OSGi-based framework to the devices or external software systems. They must be developed by hand, since they deal with technology-dependent issues. If any device or external software system was used in a previous system, the same driver can be reused.
4. The transformation engine is applied to the PervML specification. Many Java files and other resources (Manifest files, etc.) are automatically generated as a result of this action.
5. The Java files are configured in order to use the selected drivers. This configuration only implies to set up the drivers identifiers.
6. Finally, the generated files are compiled, packaged into bundles (JAR files) and deployed in the OSGi server with the implementation framework and the drivers.

In [5], a case study where this model driven approach is applied is introduced.

3. Technological Background

Until a few years ago, it was hard to find solid tools that provide facilities for developing complete support to MDSD methods, since most of the them were focused on one concrete technique for one concrete concern (metamodel definition, model management, graphical editors construction, model-to-model or model-to-text transformations, etc.). Hopefully, a new generation of frameworks and technologies which support the main steps in MDSD methods is rising. In this work, the Eclipse platform and several of its available plug-ins have been used to develop such a kind of MDSD tool for pervasive systems development. Next, a brief description of these technologies is included.

3.1. The Eclipse Platform

Eclipse was initially the IBM IDE for Java development, which was released as free software. Currently, it is the base platform for many other technologies and projects due to its very powerful modular structure and its open nature. Most of the Eclipse plug-ins are related to software development but not necessarily using Java.

Eclipse is organized in a set of first level thematic projects which guides the evolution of more concrete projects. The Eclipse Modeling Project is the first level project that unifies the modeling related projects and plug-ins that are developed by the Eclipse community (other modeling plug-ins are developed by third parties, since Eclipse is an open platform). Several of the projects in the Eclipse Modeling Project have been used for the development of our tool, like the Eclipse Modeling Framework (EMF), the Graphical Modeling Framework (GMF) and the MOFScript Tool.

3.2. The Graphical Modeling Framework (GMF)

The Graphical Modeling Framework (GMF)[2] provides a generative component and runtime infrastructure for developing graphical editors based on other two Eclipse plug-ins: EMF and GEF. The Eclipse Modeling Framework (EMF) is a modeling set of tools and code generation facilities for specifying metamodels and managing (creating / editing / saving / loading) models instances. The Graphical Editing Framework (GEF) provides libraries and a predefined programming architecture for building graphical editors using the Eclipse infrastructure.

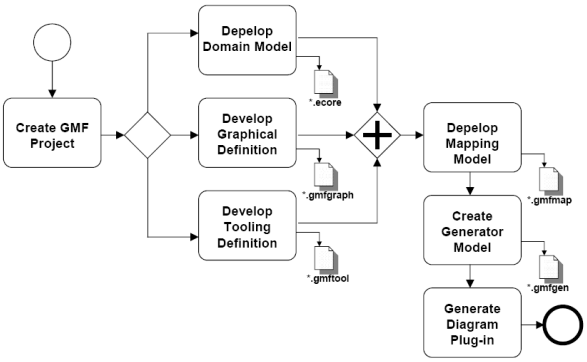

Figure 2: GMF Process Overview.

[9] provides a description of the framework and the overall process workflow that must be carried out to use the assets that are supplied by the project, which is summarized in Figure 2 that has been taken from the official tutorial. In short, the graphical editor developer must define (1) the domain model (metamodel), (2) the graphical definition (available figures), (3) tooling definition (UI related issues) and (4) the mapping model which relates the three previous models. Then, GMF is in charge of producing a generator model which can be fine tuned. With all these models, the generative component of GMF automatically produces the graphical editor.

3.3. The MOFScript Language/Tool

The MOFScript tool is included in the Generative Modeling Technologies (GMT)[3] Eclipse project. The objective of this project is "*to produce a set of research tools in the area of MDE (Model Driven Engineering*)". In this context, the MOFScript project "*aims at developing tools and frameworks for supporting model to text transformation*". This subproject has been developed in a development community at SINTEF, supported and tested by the European Integrated Project MODELWARE.

The MOFScript tool is an implementation of the MOFScript model to text transformation language [7]. This language was submitted to the OMG as response to the "*MOF Model to Text Transformation Language RFP*" [6]. It provides mechanisms for generating text from MOF-based models, controlling the sequence of execution, string manipulation, easy production of files,

[2] http://www.eclipse.org/gmf/

[3] http://www.eclipse.org/gmt/

traceability of models and generated text, etc. and it is based on the QVT standard.

4. PervGT: The PervML Generative Tool

Model driven methods must be supported by tools in order to be applicable in an effective way. The PervML Generative Tool (PervGT) allows pervasive systems developers the creation of graphical diagrams and the automatic translation of these diagrams into the final implementation code using a transformation engine.

In section 4.1., the tool is analyzed from an architectural perspective, where the relevant building blocks of the tool are identified and explained, whereas in section 4.2., the tool is described from the end users perspective and the features of the tool to improve the usability are remarked.

4.1. Tool Architecture

PervGT gives support to the most relevant aspects of a model driven generative tool:

- Model management: The models are manipulated (create / edit / save / load) conforming to the PervML metamodel. Furthermore, models should be stored according to any standard in order to improve the interoperability with others tools.
- Graphical model edition: To give full support to the modeling language the tool must represent the model elements according to a concrete syntax of the language.
- Code generation: Models are used as input to a transformation engine which produces as output source code of a pervasive system.

Figure 3 graphically shows the principal building blocks of the tool and how are they related. Next, every tool building block is briefly described.

4.1.1 Model management

The PervGT tool provides support to the creation and edition of PervML model instances conforming to the PervML metamodel. The PervML metamodel determines which model elements can be created and how they can be related. The PervML metamodel defines fourty five metaclasses and eighty six metarelationships.

To implement the PervML metamodel we have created an Ecore model. Ecore is the language that is provided by the EMF plug-in to define metamodels.

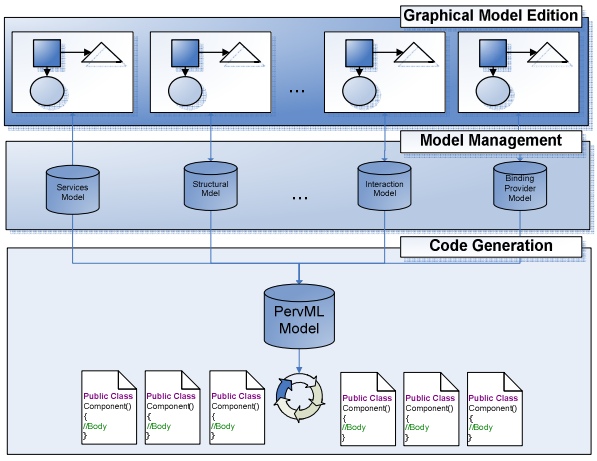

Figure 3: Tool Architecture.

The primitives that this language provides are a subset of the modeling concepts that are present in MOF 1.4. From the PervML Ecore model the plug-in EMF generates a set of Java classes representing each one of the PervML metamodel concepts. Moreover, the generated Java classes provide methods to modify PervML models according to the PervML metamodel.

To save/store the PervML model instances PervGT makes use of the EMF runtime persistency capabilities. The model persistency is achieved according to XMI 2.0. XMI is the OMG standard for interchanging models.

4.1.2 Graphical model edition

The PervGT graphical editor has been developed using the GMF Eclipse plug-in. The GMF plug-in provides an editor to specify in a declarative way graphical editors, and also provides a runtime where common functionality related to graphical editors is already implemented, like model printing or automatic layout algorithms.

The PervML language proposes different models for each one of the parts of the pervasive system that is going to be specified. According to this, the PervGT tool provides a graphical editor for each one of the different models that are proposed in the PervML. Representing model elements with graphics metaphors makes that new attributes appears those attributes associated to the model elements, like the size, position or color of the figure. To store all this new attributes the GMF plug-in provides two approaches:

1. The information that is related to the model is stored in one file (model file) and the information that is related to the graphical

representation is stored in another one (diagram file).
2. Both the model information and the graphical representation information are stored in the same file.

In the PervGT tool the persistency is realized according to the first option in order to promote separation of concerns.

Since the PervML language proposes different models to specify a pervasive system, elements in one model can be referenced form others models. In order to support this characteristic, PervGT provide facilities to refer elements from one diagram to another, guarantying the consistency between diagrams. Even the referenced elements can have distinct graphical representation in each one of the diagrams that they appear. For instance, a service could be represented in a diagram as a UML class/interface (where its operations are shown), and in another diagram the same service is represented using the "lollipop" notation.

In the process of editing a model there are some constrains that must be guaranteed. In PervGT there are defined three constraint validation levels, depending on the feedback that should be given to the end user:

1. Some actions are forbidden to be carried out by the users. For instance the PervML language only allows relationships between services. The tool shouldn't allow the creation of a relationship from a Service and other kind of element. This kind of constraints is guaranteed by construction and they don't provide feedback to the user due that they can not be violated.
2. Some actions performed by the user can incur in a temporal infraction, like creating a new service and lefting temporally the name blank. In these cases the tool do not warn the user about that situation, because it's probably that the next action will be to enter the name to the Service. To check these constraints the tool provides a validation under demand engine to avoid bothering the user with feedback messages while he is working.
3. Finally, some actions should never be allowed but they can't be avoided by the tool. For instance two services shouldn't share the same name but users can create two Services with the same name. In that case the tool cannot know which one of them is wrong. In this kind of constraints the tool checks the constraints online and it notifies users immediately.

All constraints of the diagrams are defined in the GMF plug-in using OCL expressions. An example of constrain could be the following:

```
(Defined over the metaelement Service)
Not(Self.Name.equals('PervMLService'))
```

This constraint is of the third kind, that is to say, that they are validated online and that it notifies users immediately about the existence of a service that has been named using the reserve word PervMLService.

In order to build a complete PervML model, the graphical diagrams must be completed with a textual specification. PervML uses the UML Action Semantics Language (AS) in order to specify part of the system behaviour. Due to that ASL doesn't proposes a standard concrete syntax, PervML uses a subset from the Kennedy Carter proposal [10]. The tool provides an editor of textual models where users define ASL expressions. This editor of textual models has been developed using the Xtext framework from the openArchitectureWare (oAW) plug-in. oAW is a generator framework that provides facilities to create text editors, checks constraints and has strong support of EMF.

4.1.3 Code generation

Once the pervasive system is modelled, the transformation engine can be applied to generate the code. For this task, we have used the MOFScript language which provides capabilities navigating models, creating files, etc. MOFScript takes as input one model (or several) and applies over one selected metaelement a contextual rule. The applied rule can access the element properties, navigate over the related model elements and invoke other rules.

As PervML has several models, when edition is finished, the model is divided in several files. In order to facilitate the translation process, before generating the code, PervGT joins in one unique model all the models needed to specify the pervasive system.

The code generation process must navigate through model elements and generate the needed code. For achieving this goal, three sets of rules have been implemented:

1. First, a rule navigates from the root metamodel element to the different meta-elements that are the source for creating the files to be generated.
2. The previous rule invokes a set of rules that (1) setup the environment of the meta-elements that are interesting from the code generation point of view and (2) invoke the rules that generates the text (which are described below). The

configuration of the environment implies defining files paths, files names and/or files extensions, creating the files, defining configuration parameters, therefore these rules mainly composed by properties definitions and file statements for creating the files that will be filled by the followed rules.

3. Finally, a set of rules that generate the text that is printed in the files which were created before have been implemented. In order to achieve this goal, the rules use the properties of the context meta-element where it was applied and it also can navigate over the neighbors elements for accessing their properties. These rules are mainly composed by escaped output (or print sentences), since they contain mostly text that must be copied "as is" to the target file. This text is completed by sentences that access the metaelements properties.

Next we show an example of the rules that have been defined to transform the primitive `Component` (which is defined in the PervML language) into one of the classes that it generates. Partial sections of the rules are going to be shown to illustrate the principal differences of the three kinds of rules defined above.

In the first kind of rule, the **navigation rules**, the model is navigated in order to reach all the Component instances.

```
pervml.PervMLModel::main() {
...
if ( generateComponents == true ){
   self.StructuralModel.Component
        ->forEach(c:pervml.Component){
        c.generateFromComponent()
}
}
...
}
```

Due to the `Component` primitive is a first level concept in the PervML language, the `Component` instances are reached in a near straight way from the model root element. For each one of the `Component` instances reached, a contextual rule for setting the environment is invoked.

In the second kind of rules, the **rules that setup the environment**, all the variables needed in the generation rules are initialized.

```
pervml.Component::generateFromComponent
(location:String) {

  property packageDir ="src/pervml/components"
  property baseDir = "Components/"
  property projectName = "Comp_"+ self.alias
  property projectRoot = baseDir + projectName
  property dir =
   projectRoot+"/"+packageDir+"/"+ self.alias

  var counter = 1

  /** Generating the Component **/
  file componentFile (dir+"/Component.java")
  self.generateComponent(location)

  /** Generating the Component Triggers**/
  self.ComponentTrigger->forEach
  (trig:pervml.Trigger) {
    file triggerFile (dir+"/Trigger"+counter
    +".java")
    trig.generateTrigger(counter,self)
    counter = counter + 1
  }
  ...
}
```

Once all the variables are correctly initialized the rule that configures the environment is ready to invoke the generation rules to obtain the Java code.

In the third kind of rules, **text generation rules**, `Component` properties and literal text are combined to produce the Java classes. Furthermore some `Component` model elements neighbors, like the `Trigger` element or the `Method` element, are visited in the process of generating the Java code.

```
import "SharedRules.m2t"

texttransformation Component
(in pervml:
"http:///org/oomethod/pervml.ecore") {

pervml.Component::generateComponent(){
var idCounter:Integer = 1


<%package org.pervml.application.components.%>
self.alias
<%;import org.osgi.framework.BundleContext;%>
self.genetareDependencyImports()

<%public class Component extends
org.pervml.application.services.%>
self.serviceProvided.name<%.GenericComponent {
public Component(BundleContext c, String
componentPid){
super(c,componentPid);%>

self.ComponentTrigger->
forEach(trig:pervml.Trigger) {
  println("\t\t this.listaTriggers.insertar(
  new Trigger"+idCounter+"(this));")
  idCounter = idCounter + 1
}
<%}//End Component()

public void initializeProperties(){
  this.serviceName="%>
  self.alias<%";
  this.location="%>self.getLocation() <%";
}%> //End initializeProperties()

self.ComponentFunctionalSpecification.Method
->forEach(me:pervml.Method) {
  print("\t protected ")
  me.ServiceOperation.generateOperation(
   "Implementation_")
  println("{")
```

```
  if (me.ServiceOperation.returnValue.name
     != "void")
      println("\t\t"+me.ServiceOperation.
      returnValue.name+" returnValue =" +
      me.ServiceOperation.returnDefaultValue()
      +";")
  self.generateBody()
  if ( me.ServiceOperation.returnValue.name
     != "void")
      println("\t\t return returnValue;")
      println("\t\t }")
}
<%}%> //End class Component
}//End generateComponent()
}
```

The generation rule creates the `Component` Class, indicating the genericComponent (service) of which it extends and which is specified in the models. It also adds the triggers references of the component; and then it generates the static and the dynamic part of the Component class. In other words, it generates the code that is in charge of initializing the Component properties (like the name and physical location), and the body of its methods.

As stated in Section 2, the generated files are configured in order to use the drivers that interact with the devices or software systems that implement the system services. Finally, the generated files are compiled, packaged into bundles (JAR files) and deployed in the OSGi server with the implementation framework and the drivers.

4.2. Tool User Interface

Once the tool architecture has been described, this section introduces PervGT from the end user point of view. The tool elements are going to be identified and briefly described. Furthermore the assistance capabilities of the tool are going to be analyzed.

In Figure 4, a general view of the tool is shown. The interface of PervGT is divided in seven parts:

- On the zone 1 we can see the **menu bar**, where we can find out all of the functionality that users can execute. In this menu bar, there is the *PervML Model menu* where entries to specify and validate the system and generate the code have been included.
- Below, in the zone 2, the **tool bar** gives direct access to the most used functionality, like direct model storage, change zoom, automatic layout and so on.

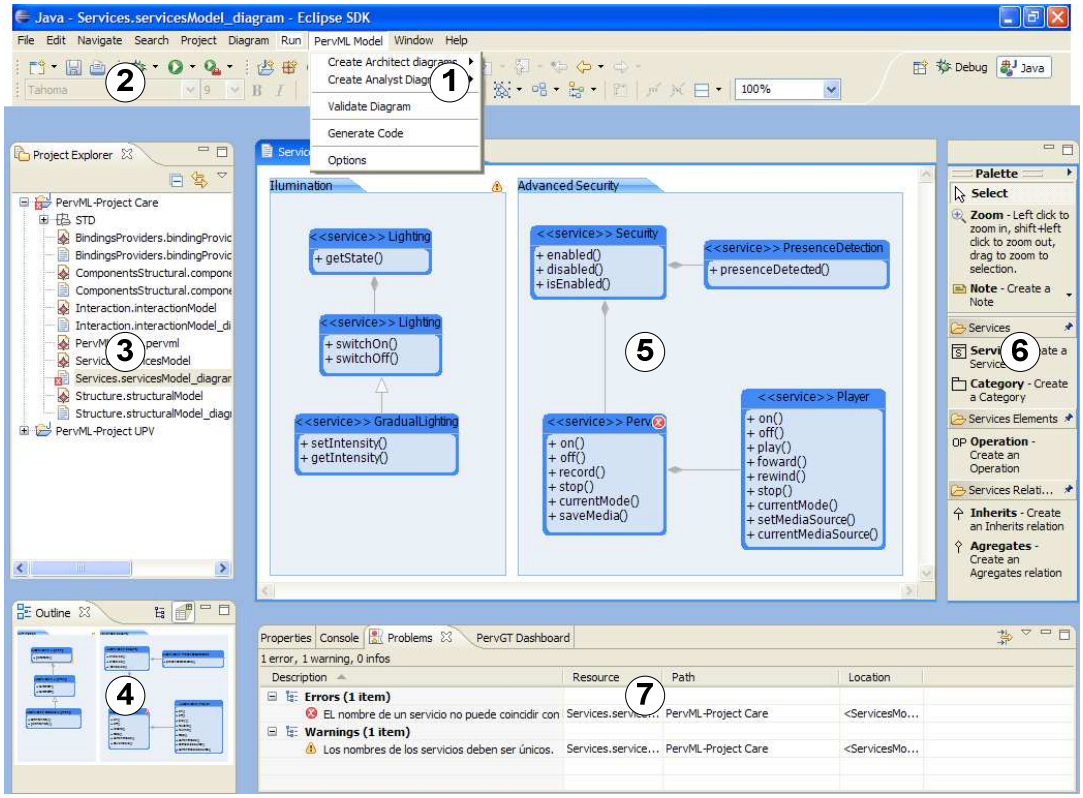

Figure 4: The PervGT user interface.

- On the left in zone 3, we can see the **Project Explorer** where projects and their contents are shown. PervGT support drag and drop an element from the Project Explorer to a diagram so it may create a link from a diagram to an element of other diagram.
- Below to the Project Explorer, in zone 4, the view **Outline** is shown. This view allows to user to have a tool global vision of the diagram and it also remarks the part of the diagram that is visualized in the main view. Besides, in this view the user can see diagram elements as a tree form.
- In the middle, zone 5, we can see the **editor area** where the model graphic representation is shown and can be manipulated.
- On the right, in zone 6, we can see the **Palette**, where there are some common functionalities and the graphical elements of the selected diagram that user can select to paint them in the diagram.
- Finally, at the bottom in zone 7, there are two tabs: the **Problem tab, which** shows the problems, or errors that have been identified in the system and the **Properties tab,** which allows users to see and to manipulate attributes of the diagram selected element.

Moreover, PervGT provides a cheat sheet for each diagram in order to guide the system specification process. A Cheat sheet is a special view that helps the user through a series of tasks to achieve an overall goal. The cheat sheets that have been created can automatically carry out certain actions for help the user. In figure 5 we show one, the tool cheat sheets for the Structure diagram.

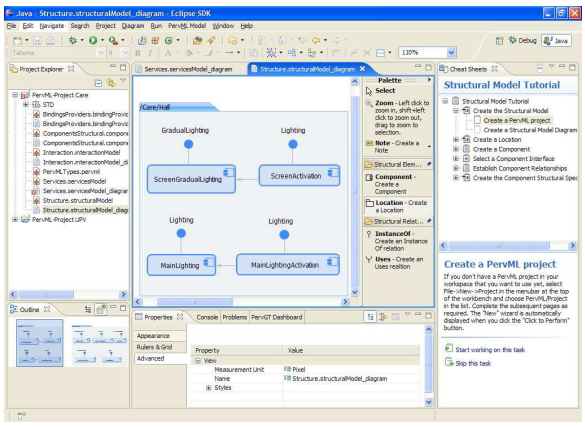

Figure 5: The cheat sheet assistant.

Furthermore the tool provides a landscape view of the complete process of specify a new Pervasive system, as it is show in the figure 6. With the help of the dashboard, the user gets feedback about the different diagrams that must be specified to completely specify a pervasive system using PervML. In the dashboard the diagrams are represented, including precedence relationships. The relationships and the information about what diagrams have been already created determinate what actions can be executed over the diagrams.

Using the elements of the dashboard, users can create new diagrams or even they can generate diagrams by default, like in the case of the State Transition Diagram (STD), where the tool can automatically generate a new STD with one unique state.

Once the complete system is specified, user can invoke the transformation engine to generate the final implementation code. From the "*PervML model*" menu, trough the "Generate" entry, the transformation engine is launched. The transformation engine generates the Java code and Eclipse projects ready to import with all the needed references to other eclipse projects. The goal of generating Eclipse projects is to facilitate to the user the Java process of manipulating and compiling the generated code.

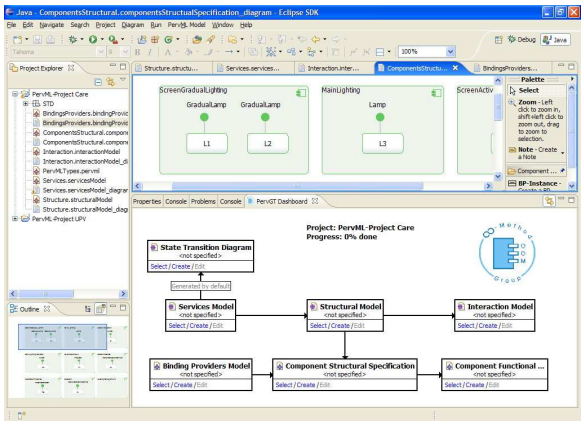

Figure 6: The dashboard view.

5. Conclusions

For making true the benefits promised by model driven approaches, tool support must be provided. The development of such a kind of tools used to be a hard task but, hopefully, big players in the software industry like Microsoft and IBM are betting strong for the model driven development. As a result of these efforts, new technologies are rising (for instance, the Eclipse GMF project and the Microsoft DSL Tools) that facilitate the development of applications which need

to manage models. In this paper, the Pervasive Generative Tools (PervGT), an Eclipse/GMF based tool, has been introduced. PervGT implements a model driven approach for the development of pervasive services in ubiquitous environments. More information about the approach can be found in [2], [3], [4] and [5]

Our future work regarding the tool is focused on two main research lines. On one hand, we plan to explore the capabilities of the method/tool for rapid prototyping pervasive systems. Since the models that are used in PervML are organized in two layers of abstraction (analysis and detailed specification), we think about providing a default detailed specification which enables the automatic code generation from the high level of abstraction models. Following this strategy, pervasive system users could early validate the abstract models.

On the other hand, we are interested on applying product line techniques on the pervasive systems field. We plan to extend the tool for supporting features diagrams which describe the most commonly characteristics of vertical domains (home, museums, retail, etc.). From this feature models, PervML models will be derived using model-to-model transformations.

In summary, we believe that model driven approaches can provide many benefits to the currently so heterogeneous and rapidly changing pervasive systems field, and these approaches must be supported by tools. PervGT is our contribution to this vision.

6. References

[1] D. Marples and P. Kriens, "The Open Services Gateway Initiative: An Introductory Overview," *IEEE Communications Magazine*, vol. 39, no. 12, pp. 110–114, 2001.

[2] J. Muñoz, V. Pelechano and J. Fons "Model Driven Development of Pervasive Systems" *I International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES 2004)*, Turku Centre for Computer Science, 2004, pp 3 - 14

[3] J. Muñoz and V. Pelechano, "Building a Software Factory for Pervasive Systems Development" *Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, Porto, Portugal, June 13-17, Springer-Verlag GmbH, 2005, 3520, pp. 329-343

[4] J. Muñoz and V. Pelechano, "Applying Software Factories to Pervasive Systems: A Platform Specific Framework" *8th International Conference on Enterprise Information Systems (ICEIS 2006)*, INSTICC Press, 2006, pp. 337-342

[5] J. Muñoz, V. Pelechano and C. Cetina, "Implementing a Pervasive Meetings Room: A Model Driven Approach" *Proceedings of the 3rd International Workshop on Ubiquitous Computing, IWUC 2006*, INSTICC Press, 2006, pp. 13-20

[6] Object Management Group. "MOF Model to Text Transformation Language - Request For Proposal" 2004

[7] J. Oldevik, T. Neple, R. Gronmo, J. Aagedal, and A. Berre, "Toward Standardised Model to Text Transformations" *Model Driven Architecture. Foundations and Applications: First European Conference, ECMDA-FA*, Springer Berlin / Heidelberg, 2005, 3748, pp. 39 - 253

[8] R. Sharp, "Deploy or die: A choice for application-led ubiquitous computing research," in *Workshop on What makes for good application led research in ubiquitous computing?*, 2005.

[9] A. Shatalin, A. Tikhomirov, "Graphical Modeling Framework Architecture Overview", *Eclipse Modeling Simposium 2006.*

[10] Ian Wilkie, Adrian King, Mike Clarke, Chas Weaver, and Chris Rastrick. UML ASL Reference Guide. Kennedy Carter Limited, 2001