VML4RE User Manual

FCT/UNL

1 Contents

Introduction

Managing variability in Requirements Engineering (RE) is a key challenge in Software Product Line (SPL) engineering. An important part of variability management is the ability to express explicitly the relationship between variability models (expressing the variability in the problem space, for example using feature models) and other artefacts of the product line, for example, requirements models. Once these relations have been explicitly represented, they can be used for a number of purposes, most importantly to automatically derive product instances based on product-configuration specifications, but also for other purposes such as trace-link generation and consistency checking of SPL models.

In this document we present the user guide for the VML4RE tool suite. We illustrate its use for UML requirements models in a home automation case study called Smart Home. The eclipse project containing the application of VML4RE to Smart Home can be found here.

2 System Requirements and Installation

The VML4RE tool was developed as an Eclipse plug-in and was designed to work with the following set of requirements:

- Java SE Runtime Environment (JRE) 6
- Eclipse Modeling Tools Bundle v.3.4.1
- OpenArchitectureWare v.4.3.1 [1]
- UML2 Tools for Eclipse [2]
- Feature Modeling Plug-in (FMP) v.0.7 [3]
- ATF v0.2.1 (All ATF requirements must also be satisfied. Please, check ATF documentation)

Since VML4RE is an instantiation of the VML* Suite to deal with use cases and activity models, the suite should be included in your eclipse installation. VML* can be downloaded from [4].

To install VML4RE plug-in, copy the following files to the directory *dropins* your Eclipse installation:

- unl.vml4req_1.1.1.jar
- org.ample.vmlstar.util_1.1.1.jar
- org.ample.vmlstar.langinst_1.1.1.jar
- org.ample.vmlstar.langinst.model_1.1.1.jar

- org.ample.vmlstar.langinst.model.edit_1.1.1.jar
- org.ample.vml.vml4req_1.1.1.jar
- org.ample.vml.vml4req.generator_1.1.1.jar
- org.ample.vml.vml4req.editor_1.1.1.jar

Note that the numbers in the names can change, depending on the current version of the tool.

This tool can also be found already installed and ready to use in the AMPLE Eclipse Bundle, which can be downloaded from here [5].

3 Available Actions in VML4RE

VML4RE is an instance of the VML tool suite. In order to create the VML4RE language, a set of actions specific for use cases and other requirements models were implemented. For more information about the process of instantiating the VML* framework, please consult the VML* documentation which can be obtained from the AMPLE website. These actions can then be used to manage variability, i.e., when creating a VML4RE specification as we will see in this document.

Table 1 shows the Set of available actions for use case and activity models.

Action Signature	Description
`insertUseCase` (String name, Package p)	A new use case named `name` is inserted into package `p`.
`insertPackage` (String name, Package p)	A new package named `name` is inserted into package `p`.
`createInclude` (List[UseCase] from, List[UseCase] to)	A new dependency of type `<<include>>` is created between each of the source use cases and each of the target use cases.
`createExtends` (List[UseCase] from, List[UseCase] to)	A new dependency of type `<<extends>>` is created between each of the source use cases and each of the target use cases.
`createInherits` (List[UseCase] from, List[UseCase] to)	A new dependency if type `<<inherits>>` is created between each of the source use cases and each of the target use cases.
`createAssociation` (List[UseCase] from, List[UseCase] to)	A new association relationship is created between each of the source model elements and each of the target models elements.
`insertActor` (String name, Package p)	A new actor named `name` is inserted into package `p`.
`insertPackage` (String name, Package p)	A new package named `name` is inserted into package `p`.
`removeElement (Element elem)`	The element `elem` is removed from the model.
`createActivityModel` (String name, Package p)	A new activity model named `name` is inserted into package `p`.
`createAction` (String name, Package p)	A new opaque action named `name` is inserted into package `p`.

`createObjectAction` (String name, Package p)	A new object action named name is inserted into package `p`.
`connectActivityElements` (ActivityNode source, ActivityNode target, String guard)	The activity elements `source` and `target` are connected through a control flow with guard `guard`. If `guard` equals "", no guard name is created.
`createDecisionNode` (String name, Package p)	A new decision node named `name` is inserted into package `p`.
`createActivityParameter` (String name, Package p)	A new activity parameter named name is inserted into package `p`.
`createOutputPin` (String name, OpaqueAction oa)	Creates a new output pin named `name` for the action `oa`.
`connectPins(OpaqueAction oa1, String pin1name, OpaqueAction oa2, String pin2name ):`	Connect pin named `pin1name` of action `oa1` with pin names `pin2name` of action `oa2`.
`replaceActionByActivity (OpaqueAction oa, Package p)`	Replace the action `oa` with the contents present in the package `p`
`Trace (Element e)`	This operator traces the features where this action appear to the element or set of elements referenced by `e`. *It*

Table 1 - List of available actions for VML4RE

The operator — trace (Element elem) — allows to define explicitly trace links from features included in a feature expression with target model elements. To exemplify, if the vml4re specification contains the following variant:

```
variant for f1 {
      trace ( "refelemx");
}
```

the feature `f1` will be traced to the element referenced by `refelemx`;

4 Applying VML4RE to a Case Study

To demonstrate how VML4RE can be used, we show the application of the tool with a case study called *Smart Home* [6, 7]. Smart home is a home automation software product line case study developed in the context of the AMPLE project. Smart homes have a wide variety of electronic and electrical devices which include lights, thermostats, blinds and fire detection sensors, security devices such as cameras, white goods such as washing machines, communication devices such as phones and entertainment devices such as televisions. The Smart Home system is designed to coordinate the behaviour of the devices to fulfil complex tasks automatically. It also enables the inhabitants to visualize and control the status of the devices from a common user interface. For brevity and clarity we only use some of the features of the Smart Home.

4.1 Creating a new VML4RE Project

The first step is to create a new VML4RE project. For this, Right Click on the package explorer view of your workspace:

New-> Other… -> XText DSL Wizards -> VML4RE Project. Give a name to your project (for example, SmartHome)

Next, a new blank VML4RE project is created with the structure depicted in Figure 1.

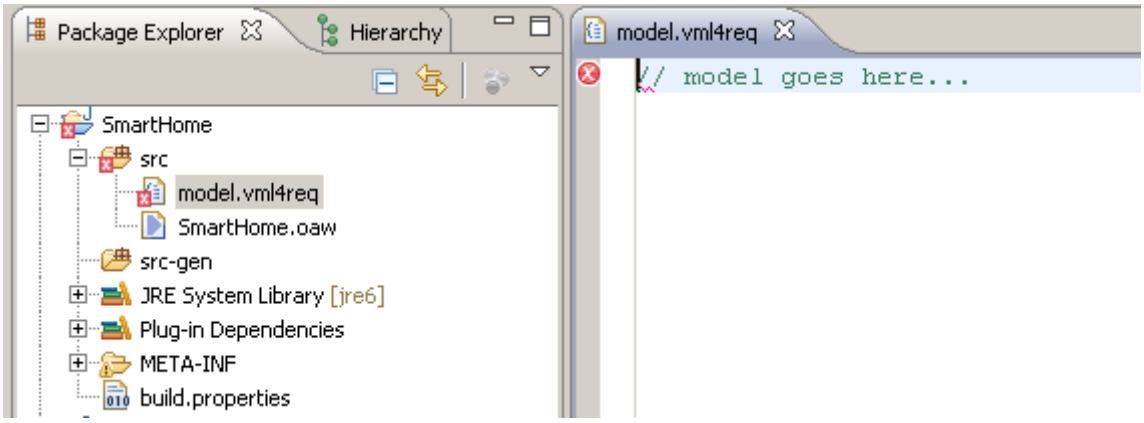

Figure 1 - Empty VML4RE project

Before creating a VML4RE specification, it is necessary to create the feature and requirements models. These steps are demonstrated in the next sections.

4.2 Modeling Commonalities and Variabilities

The next step of our approach is to model the commonalities and variabilities of the SPL. Figure 2 (left) shows the feature model of the Smart Home product line and Figure 2 (right) shows one of its possible configurations called Economic Smart Home. In the Economic edition the optional features that will not be included in the final product are not ticked. Therefore, camera surveillance and notifications via internet are not part of the final product.

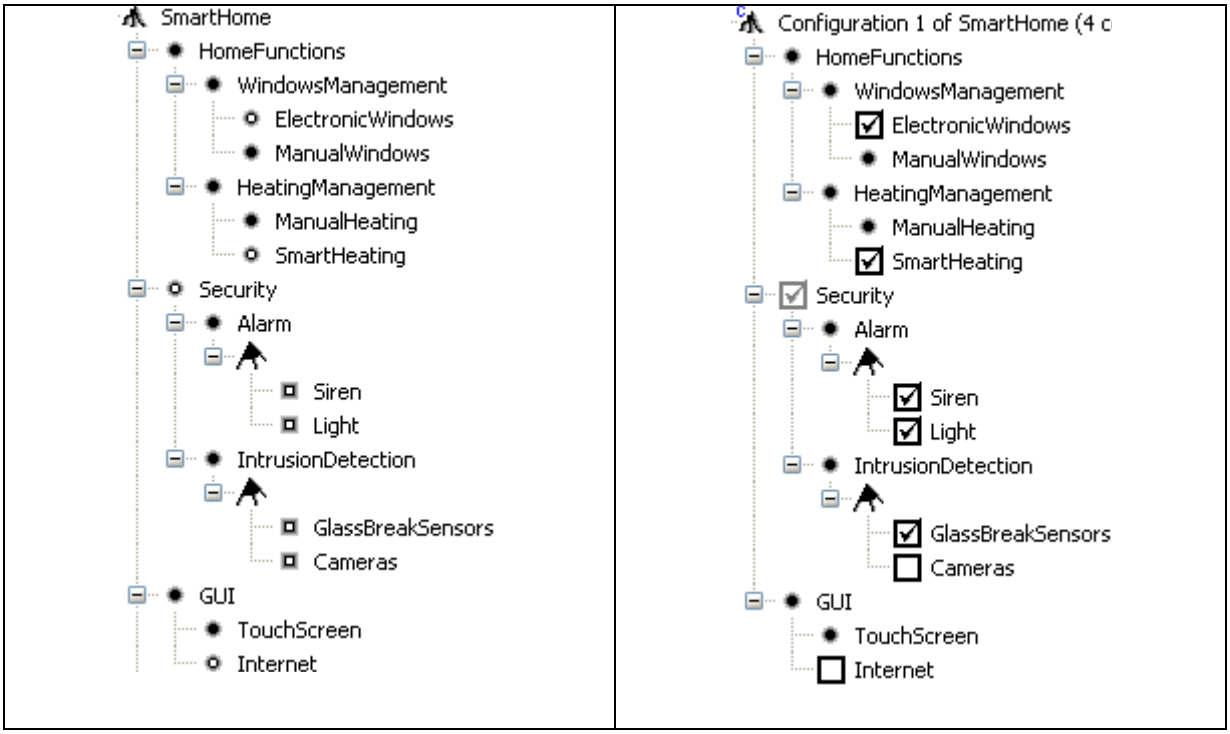

Figure 2- Feature model (left) and one of its configurations called Economic Smart Home (right).

Both models were created using the Feature Modelling Plug-in (FMP) [3] for Eclipse.

4.3 Modelling SPL Requirements Models with UML

The second step of our approach consists on modeling the requirements of the SPL using UML, such as use case and activity models. We use the UML2 plug-in for Eclipse [2] to model both use case and activity diagrams. However, the user could use other commercial tools like Magic Draw [8] for example, since VML* is general enough and allows the user to define or implement how models should be loaded.

To create use case and activity models, right click on the folder where do you want to put the new model (e.g., scr), *New -> Other… -> UML 2.1 Diagrams -> UMLUseCase Diagram*. Give a name to the new Model (*SmartHome.uml*, for example).

To create activity models you can right click on *SmartHome.uml* and then right click on *Initialize Activity Diagram*. Figure 3 shows an example of a core use case model for the Smart Home case study.

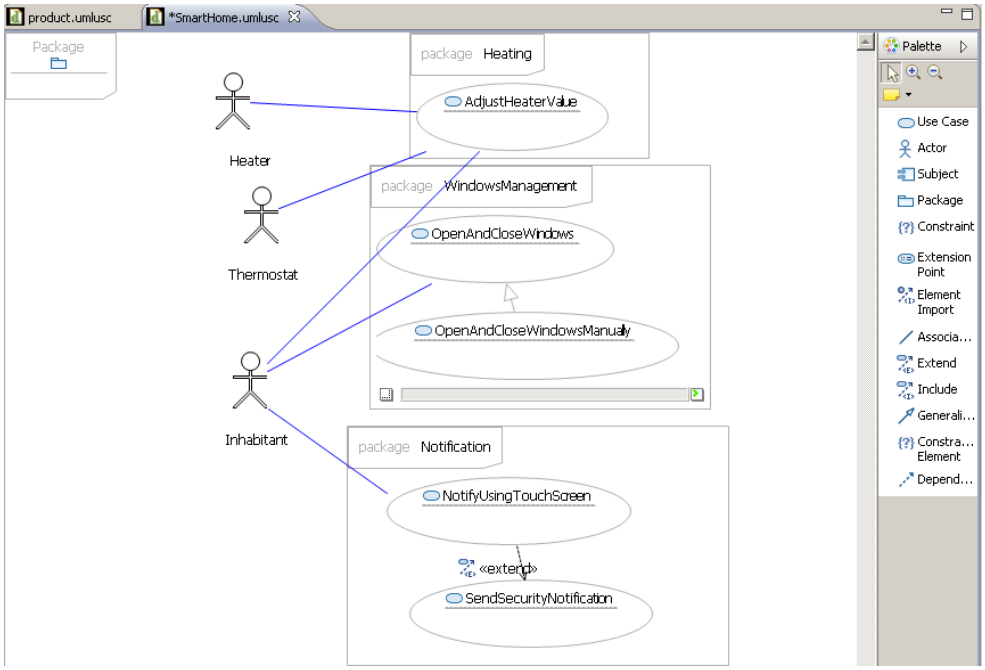

Figure 3 - Core UML Use Case Model for Smart Home

Figure 4 shows an example of the activity diagram for the scenario *Adjust Heater Value*.

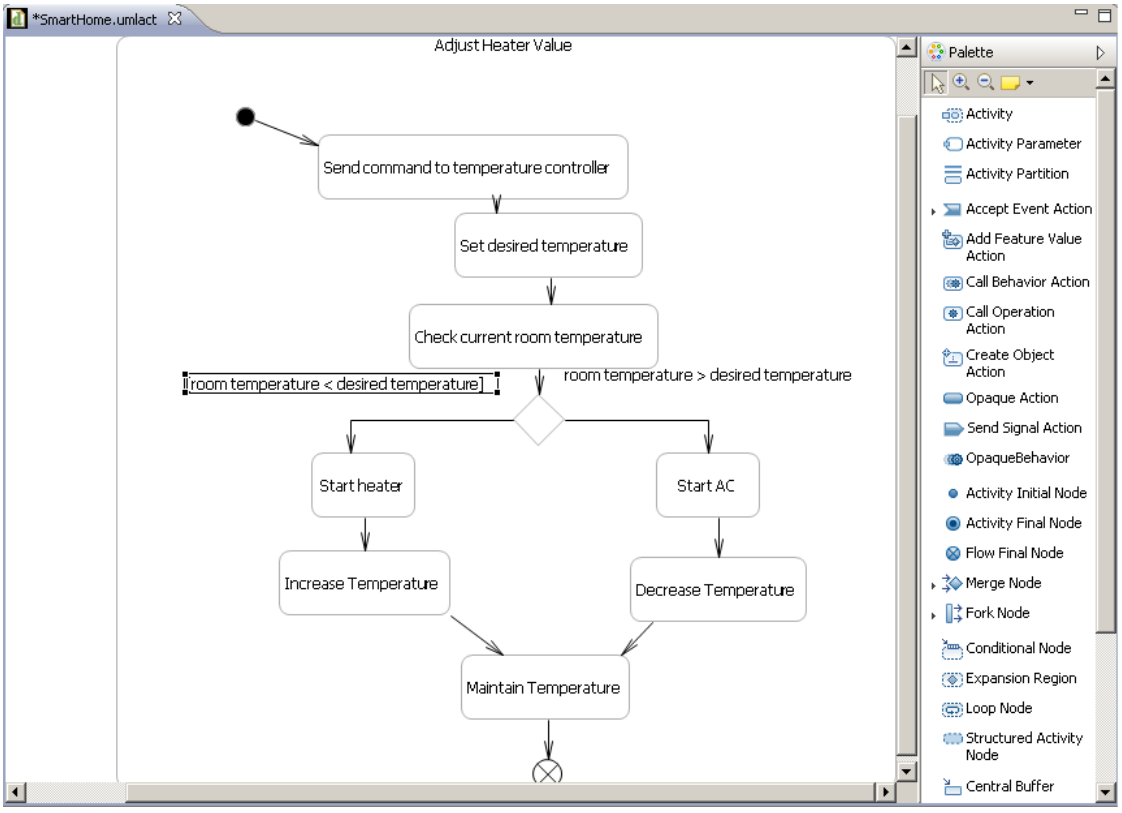

Figure 4 - Activity Model for Activate Secure Mode scenario

4.4 Writing the VML4RE Specification

At this step, we can start writing the VML4RE specification.

Table 2 summarizes the general structure of a VML* specification. More information can be found in the VML Suite documentation available in the AMPLE website.

VML Block	Description
Import features path	This block imports the feature model located in the path to be used along the vml4re specification.
Import core path	This imports the core model to be used along the vml4re specification.
atf repname	This line, when present, creates a new trace link repository in ATF. Trace links generated during the execution of this vml4re specification will be persisted in the created repository.
Variant v_name feature_expression	Variant allows giving a name to a feature expression. A feature expression defines logic between a set of features. Inside a variant, a set of actions should be defined.

Table 2- General structure of a VML specification

For demonstration purposes, simply copy the following vml4re specification into the vml4req file. Note that the first two lines should be adapted, according to the path where the previously created models are located. In the following example, the models are in *C:/VML4RESmartHome/*.

```
import features <"File:C:/VML4RESmartHome/SmartHome.fmp">;

import core <"File:C:/ VML4RESmartHome/SmartHome.uml">;

atf "c:/tmp"

concern SmartHome {
        variant for GUI {
                trace("Notification");
        }

        variant for WindowsManagement {
                trace("Windowsmanagement");
        }

        variant for HeatingManagement {
                trace("Heating");
                trace("Thermostat");
                trace("Heater");
        }

        variant for SmartHeating {
                insertUseCase ( "ControlTemperatureAutomatically", "Heating");
                insertUseCase ( "CalculateEnergyConsumption", "Heating");
                createInclude ( "Heating::ControlTemperatureAutomatically" ,
                "Heating::AdjustHeaterValue" );
                insertActor ( "WindowActuator", "");
                createAssociation ( or ("WindowActuator", "Thermostat") ,
                "Heating::ControlTemperatureAutomatically");
                createExtends ( "Heating::ControlTemperatureAutomatically",
                "Heating::AdjustHeaterValue" );
                createExtends ( "Heating::CalculateEnergyConsumption",
                "WindowsManagement::OpenAndCloseWindows" );
        }

        variant for ElectronicWindows {
                insertUseCase ( "OpenAndCloseWindowsAutomatically", "WindowsManagement");
                insertActor ( "WindowActuator", "");
```

```
                insertActor ( "WindowSensor", "");
                createInherits ( "WindowsManagement::OpenAndCloseWindowsAutomatically" ,
                "WindowsManagement::OpenAndCloseWindows");
                createInclude ( "Heating::ControlTemperatureAutomatically",
                "WindowsManagement::OpenAndCloseWindowsAutomatically" ) ;
                createAssociation (or ( "WindowSensor","WindowActuator")
                ,"WindowsManagement::OpenAndCloseWindowsAutomatically" );
        }

        variant secure_it for Security {
                insertPackage ( "Security" , "");
                insertUseCase ( "SecureTheHouse" , "Security" );
                insertUseCase ( "ActivateSecureMode" , "Security" );
                createAssociation ("Inhabitant", "Security::.*");
                createInclude ( "Security::SecureTheHouse",
                or("Notification::SendSecurityNotification","WindowsManagement::OpenAndCloseWin
                dowsAutomatically"));
                createActivityModel ( "ActivateSecureMode", "" );
                createAction ("VerifyInstalledGlassBreakSensors", "ActivateSecureMode");
                connectActionWithInitialNode
                ("ActivateSecureMode::VerifyInstalledGlassBreakSensors", "ActivateSecureMode");
                createAction ("WaitForAlarmSignal", "ActivateSecureMode");
                connectActionWithFinalNode ("ActivateSecureMode::WaitForAlarmSignal",
                "ActivateSecureMode");

                connectActions("ActivateSecureMode::VerifyInstalledGlassBreakSensors","Activate
                SecureMode::WaitForAlarmSignal" );
                }

        variant for Siren {
                insertActor ( "Siren", "" ) ;
                createAssociation ("Siren", "Security::SecureTheHouse" );
        }

        variant for Light {
                insertActor ( "Lights", "" ) ;
                createAssociation ("Lights", "Security::SecureTheHouse" );
        }

        variant for GlassBreakSensors {
                insertActor ( "GlassBreakSensor", "" ) ;
                createAssociation ("GlassBreakSensor", "Security::SecureTheHouse" );
                createAssociation ("GlassBreakSensor", "Security::ActivateSecureMode" );
        }

        variant for Cameras {
                insertActor ( "Cameras", "" ) ;
                createAssociation ("Cameras", "Security::SecureTheHouse" );
        }

        variant network for Internet {
                insertUseCase ( "NotifyUsingInternet" , "Notification" ) ;
                createExtends ( "Notification::NotifyUsingInternet" ,
                "Notification::SendSecurityNotification");
        }
}

order (network, secure_it);
```

Figure 5 - VML4RE Smart Home Specification

4.5 Deriving Product Models

The product models are generated when the vml4re specification is compiled and executed. To compile the vml4re specification, right click on the vml4re file and select:

Vml4req -> compile

When this operation is concluded, to execute the specification, right click again on the vml4re file and select:

Vml4req -> configure

Here you should select the file that contains the configuration of the feature model. In this case, since the FMP files already persists feature models and configurations in the same file, you should select the same feature model file you defined in section import of your vml4re specification.

After selecting the configuration file, some internal actions are performed. Finally, inside the folder */src-gen* of your project you can find the product model.

Considering the feature model and its configuration (presented in Section 4.1) and the vml4re specification (Section 4.4), the product use case model was generated. Figure 6 depicts the generated product use case model for the configuration.

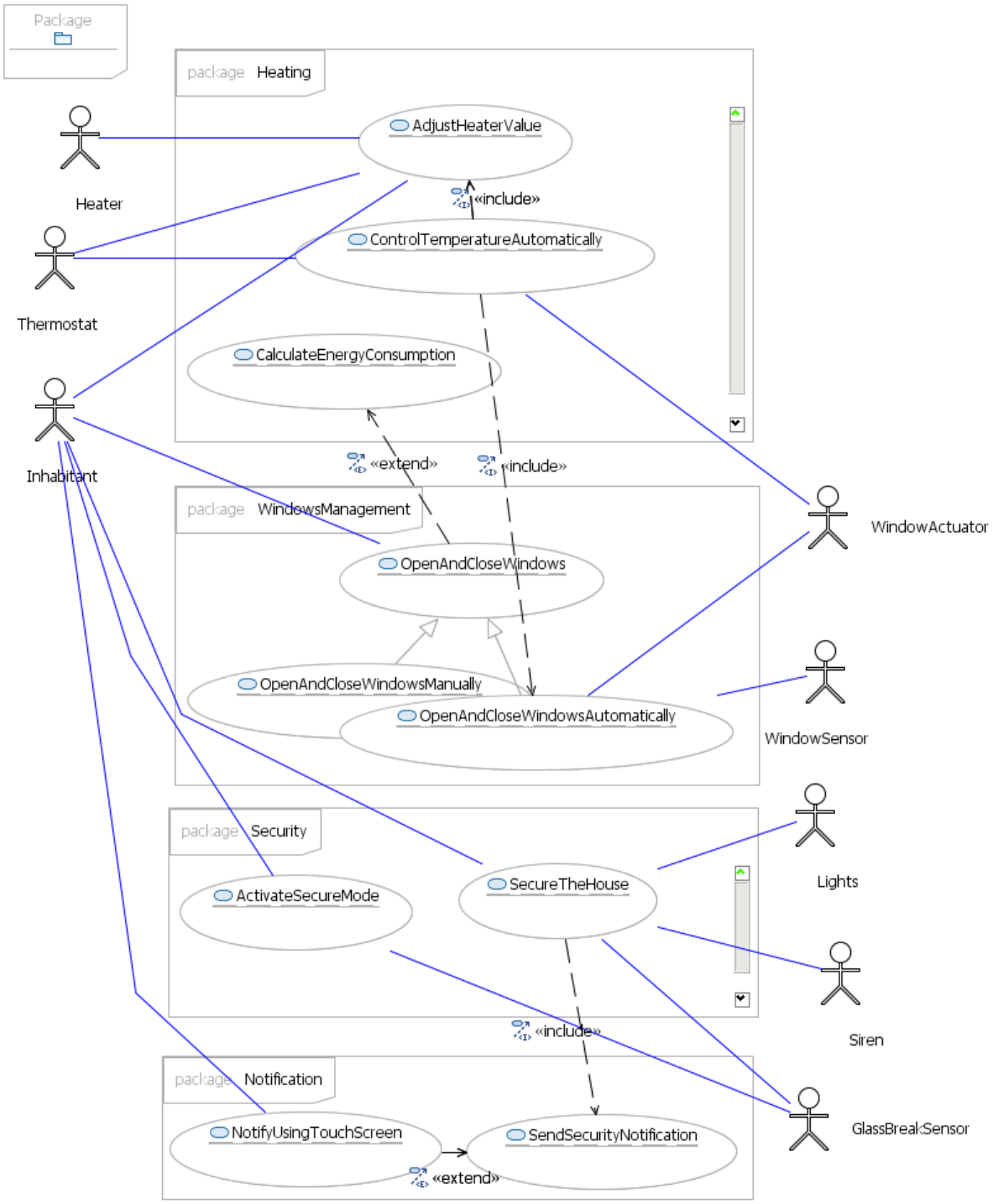

Figure 6 - Product use case model for Economic Smart Home

Figure 7 depicts the resulting product activity diagram for the scenario *Activate Secure Mode*.

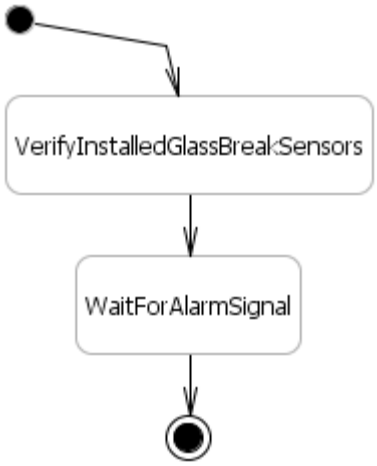

Figure 7 - Activity Model for optional Scenario Activate Secure Mode

4.6 Viewing Generated Trace Links

To view the generated trace links, it is necessary to import the project with the ATF repository to the workspace. To do this, you should right click and select:

import.. -> General -> Existing Projects into Workspace -> Select root directory

Then, browse to the directory which contains the repository (in this case, inside the path *c:\tmp,* according to the vml4re specification).
After importing, the repository should be connected and a new traceability scenario should be created (see [9] for more information on how to do this). Then, the tree register should be executed. Figure 8 is a screenshot of the AMPLE Traceability Framework which allows to filter the results the user want to see, by selecting the source, target and link types.

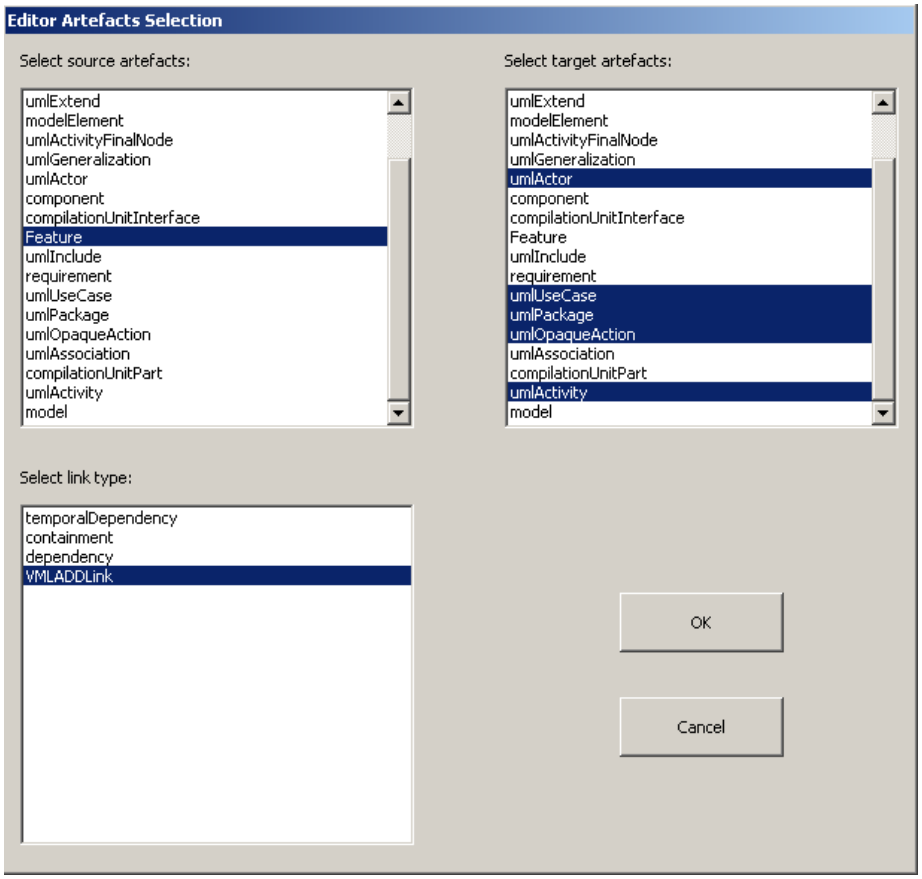

Figure 8 - Choosing artefacts and link types in ATF

For this examples, we have selected the following types:

- Source element types: *Feature*
- Target element types: *umlActor, umlUseCase, umlPackage, umlOpaqueAction, umlActivity*
- Link Types: *VMLADDLink*

Figure 9 shows part of trace links generated for the product Economic Smart Home in the ATF plug-in, for both features *HeatingManagement* and *Security*.

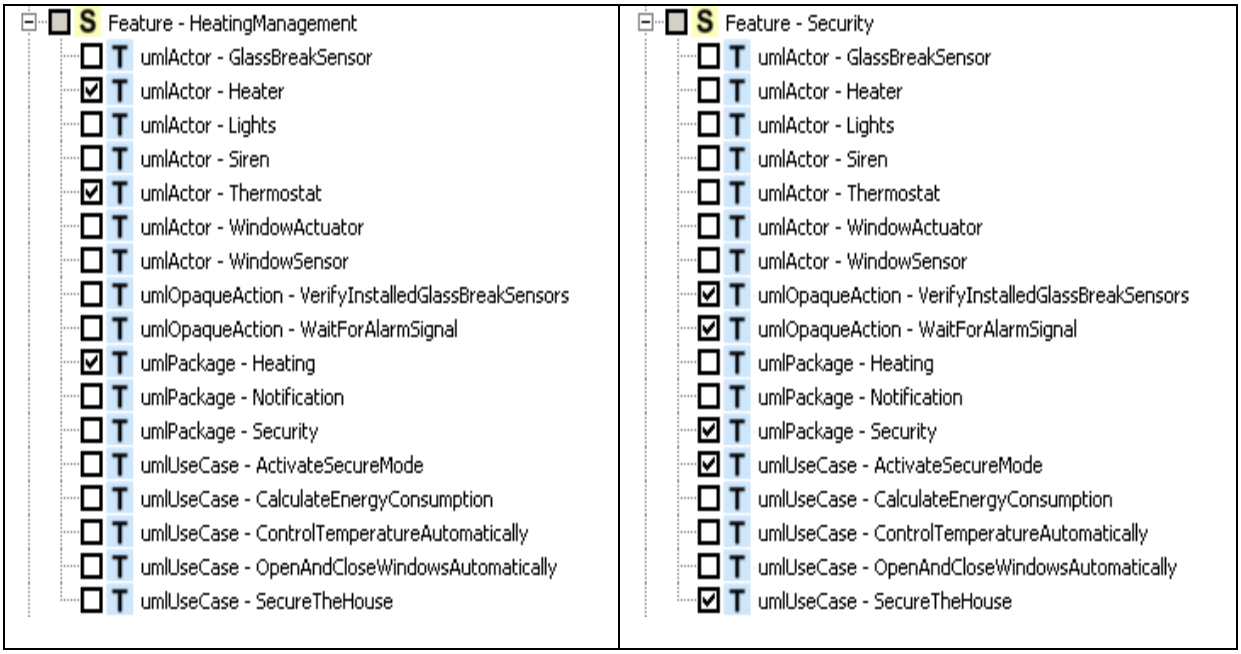

Figure 9 – Generated trace links for the features HeatingManagement (left) and Security (right)

The trace links generated for the feature *HeatingManagement* were created based on the explicit use of the operator *trace* in the vml4re specification (Section 4.4). On the other hand, the trace links generated for the feature *Security* were created based on the implicit information contained in the actions present in the vml4re specification, i.e., VML4RE identifies places where new model elements are created or existing ones deleted (we define pointcuts into our transformation implementation to support this).

References

[1] "OpenArchitectureWare," http://www.openarchitectureware.org/ , 2008.

[2] "UML2 Plugin for Eclipse," www.eclipse.org/uml2/.

[3] "Feature Modelling Plugin (FMP) for Eclipse," http://gsd.uwaterloo.ca/projects/fmp-plugin/.

[4] "VML* Download," http://www.steffen-zschaler.de/publications/vmlstar/, 2009.

[5] "AMPLE Eclipse Bundle Distribution," http://www.caesarj.org/downloads/ample/platform/ample-tools-eclipse-20090706.zip, 2009.

[6] E. Figueiredo, N. Cacho, C. Sant'Anna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. Ferrari, S. Khan, F. C. Filho, and F. Dantas, "Evolving software product lines with aspects: an empirical study on design stability," in *Proceedings of the 30th international conference on Software engineering* Leipzig, Germany: ACM, 2008.

[7] T. Young, "Using AspectJ to Build a Software Product Line for Mobile Devices - www.cs.ubc.ca/grads/resources/thesis/Nov05/Trevor_Young.pdf," University of Waterloo, 2005, p. 73.

[8] "MagicDraw," http://www.magicdraw.com/, 2009.

[9] A. Sousa, "AMPLE Traceability Framework Frontend Manual," http://ample.di.fct.unl.pt/Front-End_Framework/ATF%20Front-end%20Manual.pdf, 2008.