Enabling End-users Participation in an MDD-SPL Approach

Francisca Pérez, Pedro Valderas, Joan Fons
Research Centre on Software Production Methods
Technical University of Valencia
Valencia, Spain
{mperez, pvalderas, jfons}@pros.upv.es

***Abstract*—Developing smart home systems that properly fit end-user needs is not always an easy task due to the lack of understanding that may exist between end-users and system developers. In the context of Software Product Lines, several approaches have been presented to improve the development of smart home system functionality. However, little support is provided to improve the interaction with end-users. In this work, we extend a Software Product Line based on Model-Driven Development with an interactive design tool that allows end-users to actively participate in the SPL. This tool allows end-users to configure the decision model that drives the production process of the software product line by themselves. In order to develop this tool we have been inspired by well-known and tested end-user techniques and interaction patterns that improve the user interface usability.**

I. Introduction

Smart home systems are in charge of providing different services to support the daily activities of the inhabitants of a home. In order to do this, smart home systems automatically perform actions such as turning the lights on [1], controllling a thermostat, closing the blinds, etc. However, all these actions must be performed according to the user's preferences and needs.

Adapting smart home systems to end-users needs is not always an easy task due to the lack of understanding that may exist between end-users and system developers. End-users are the owners of the domain of knowledge, the ones with more in-depth knowledge about both the services that must be provided by the system and the environment in which the system is going to be deployed. However, many times they do not have the ability of transmitting this information properly. We think that this can be improved by providing mechanisms that allow end-users to actively participate in the development process.

In the area of Software Product Lines (SPL), many efforts have already been made to improve the development of smart home systems [2], [3]. However, these approaches focus mainly on providing developers with techniques and tools to develop the system functionality, and they pay little attention to the interaction with end-users. In this work, we face the problem of allowing end-users to actively participate in the development of a smart home within an SPL.

To do this, we have extended a Software Product Line to develop smart home systems [4], which is based on Model Driven Development (MDD). The proposed extension consists of an interactive design tool that allows end-users to create tailored solutions that directly reflect their needs and expectations. To do this, we have been inspired by well-known and tested end-user techniques and interaction patterns that improve the user interface usability [5], [6], [7].

Considering the schema of the MDD-SPL (see Fig. 1), where a product operation transforms input assets into an output system according to the configuration specified in a decision model, the contribution of this work is an end-user tool that enables end-users to configure the decision model that drives the production process by themselves.

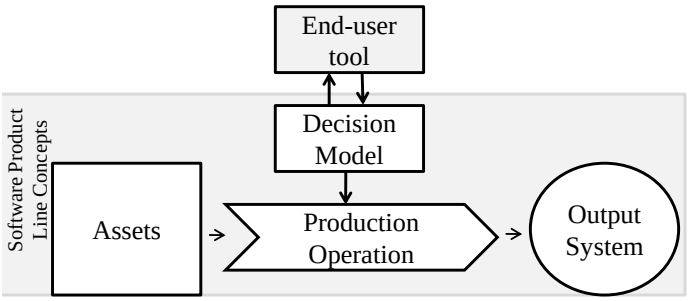

Fig. 1. Approach overview

The rest of this paper is structured in the following way: Section II presents the related work in the field of the end-user development techniques for smart homes. Section III presents the MDD-SPL for developing smart home systems. Section IV introduces the end-user tool and the interaction patterns that have been applied to improve the interface usability. Section V presents some aspects of the technology used to implement this tool. Finally, section VI concludes the paper.

II. Related work

There are several works that show how to combine MDD and SPLs [8], [2]. Voelter and Groher [2] describe an approach where development is combined with model-driven development. They define aspects at the modelling level, the transformation level, and the implementation level. They apply their approach to the Smart Home domain. Anastasopoulos et al. [8] apply a combination of both MDD and SPL to the Ambient Assisted Living (AAL) domain. They express variations in smart home functionality as features, and synthesize AAL specifications by composing features. Compared

to our work, the above approaches do not involve end-user expectations in the MDD-SPL, which is essential for the successful development of Smart Homes [9]. Other works such as [10] presents a tool to support end-users in working with large-scale product line variability models in product derivation. This tool is based on derivation models and it provides end-users with a textual visualization which allows end-users to set values on decisions by answering questions. However, the use of a visual language seems to be the best option since visual languages have demonstrated to be more intuitive and easier to be used by users than other options like textual languages [11], [12].

Many research initiatives seek to allow end-users to program or customize their systems using end-user techniques as Pervasive Interactive Programming (PiP) [13], or CAPpella [14]. Furthermore, other research initiatives allow end-users to interact with their system using metaphors as jigsaw puzzle pieces [15], or magnetic refrigerator poetry [16]. Some of these well-accepted end-user techniques are:

- **Natural Programming [17]:** it is an application of the standard user-centered design process to the specific domain of programming languages and environments. The premise of this approach is that programmers will have an easier job if their programming tasks are made more natural. For example, HANDS [18] is a programming system for children. HANDS is an event-based system featuring a concrete model for computation based on concepts that are familiar with non-programmers. The computation is represented as an agent named Handy, sitting at a table handling a set of cards.
- **Programming By Example [19]:** also called Programming by demonstration (PBD) because the user shows examples of the desired behaviour to the computer. For example, Pervasive Interactive Programming (PiP) [13] provides a platform that uses the physical user space as the programming environment providing the user with a natural and more familiar mechanism to "program" the functionality they require to suit their particular needs.
- **Visual Programming [20]:** it is the use of visual expressions in the programming process. For example, Alice [21] is an innovative 3D programming environment that allows students to learn fundamental programming concepts in the context of creating animated movies and simple video games.
- **Jigsaw metaphor [15]:** it is based on the familiarity evoked by the notion and the intuitive suggestion of assembly by connecting pieces together. Essentially, it allows end-users to make variability decisions through a series of left-to-right couplings of pieces. For example, ACCORD has developed the Tangible Toolbox [22], based on a shared Data Space, that enables people to easily administer and re-configure services based on embedded devices around the home. This toolkit also enables devices to integrate with each other through several different editors. One of these editors uses the jigsaw metaphor to create new services.

Although these techniques encourage end-users to participate in the creation of software systems, they do not address a process where end-users can specify the requirements of the system. Our approach applies end-user techniques within an MDD-SPL in order to allow end-users to actively participate in the configuration of the desired software (in this particular case, a smart home system).

III. MDD-SPL for Smart Homes

In this section, we illustrate the SPL for smart home systems. Fig. 2 illustrates the models used in the SPL. The input assets consist of a collection of models describing all smart homes that can be produced. These models are created by using the PervML language. A smart home is uniquely defined by the selections made on the feature model, which plays the role of decision model. The selected features determine which elements of the PervML models are used for the initial configuration of the smart home by means of a Realization Model. Finally, the output system is obtained through a model transformation.

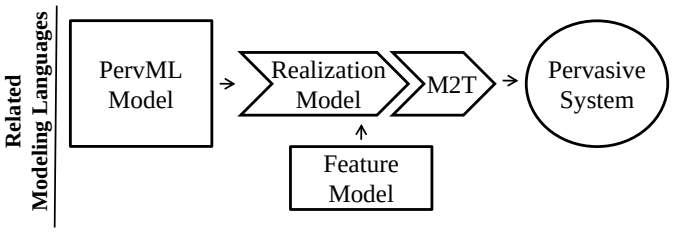

Fig. 2. MDD-SPL for Smart Homes

The following subsections provide details about the models involved in the SPL.

A. The PervML model

Pervasive Modeling Language (PevML) [23] is a DSL for describing pervasive systems using high-level abstraction concepts. This language focuses on specifying heterogeneous services in specific physical environments such as the services of a smart home. These services can be combined to offer more complex functionality by means of interactions. These services can also start the interaction as a reaction to changes in the environment. The main concepts of PervML are: (1) a *Service* coordinates the interaction between suppliers to accomplish specific tasks (these suppliers can be hardware o software systems); (2) a *Binding provider* (BP) is a supplier adapter that embeds the issues of dealing with heterogeneous technologies; (3) an *Interaction* is a description of a set of ordered invocations between Services; and (4) a *Trigger* is an ECA rule (Event Condition Action) that describes how a Service reacts to changes in its environment. This DSL has been applied to develop solutions in the smart home domain [24].

This model (see the bottom of Figure 3) describes the building blocks for the assembly of a pervasive system [23].

The grey blocks implement the functionality of the selected features. The white blocks enable an alternative functionality of the system. The (l), (o), (m) and (p) blocks provide adapters for the new resources available.

B. The feature model

Feature models are widely used to describe the set of products in a software product line in terms of features. In these models, features are hierarchically linked in a tree-like structure and are optionally connected by cross-tree constraints. There are many proposals for the type of the relationships and the graphical representation of feature models [25]. We have chosen the Feature Model [26] as the modeling language because it is feature reasoning oriented and has a good tool support [27].

This model (see the top of Fig. 3) determines the initial and the potential features of the smart home. The grey features are selected to specify a member of the smart home family. The white features represent potential variants. Initially, the smart home provides *Automated illumination, Presence simulation and a Security* system. This security system relies on *In home* detection (inside the home) and a siren alarm. The system can potentially be upgraded with volumetric presence detection and more alarms to enhance home security.

The feature model also determines how the features relate to each other by cross-tree constraints. As the feature model of Fig. 3 shows, these relationships are: **Optional** represented with a small white circle on top of the feature, **Mandatory** represented with a small black circle on top of the feature, **Multiple choice** represented with a black triangle, **Single choice** represented with a white triangle, **Requires** which it is represented with a dashed arrow and **Excludes** represented with a dashed double-headed arrow.

C. Realization model

The realization model is an extension that we have incorporated to Atlas Model Weaving (AMW) [28] in order to relate the SPL features to the PervML elements. AMW is a model for establishing relationships between models. Our extension augments the *AMW relationship* with the *default* and *alternative* tags. This augmented relationship is applied between features and PervML elements (BPs and Services). In the context of a BP, the *default* relationship means that the BP is selected for the initial configuration of the system. The *alternative* relationship means that the BP is considered a quiescent element that should be incorporated to the SPL product, but does not participate in the initial configuration. Quiescent BPs provide an alternative BP to replace the default BP in case of fault. The more quiescent BPs identified, the more flexible the adaptation will be.

This model (see the middle of Figure 3) establishes the relationships between the features and the PervML elements. For instance, the visual alarm feature is related to a BP (p) for visual alarms, but, alternatively, it can be replaced with a BP (m) that emulates the visual alarm by using the blink lighting.

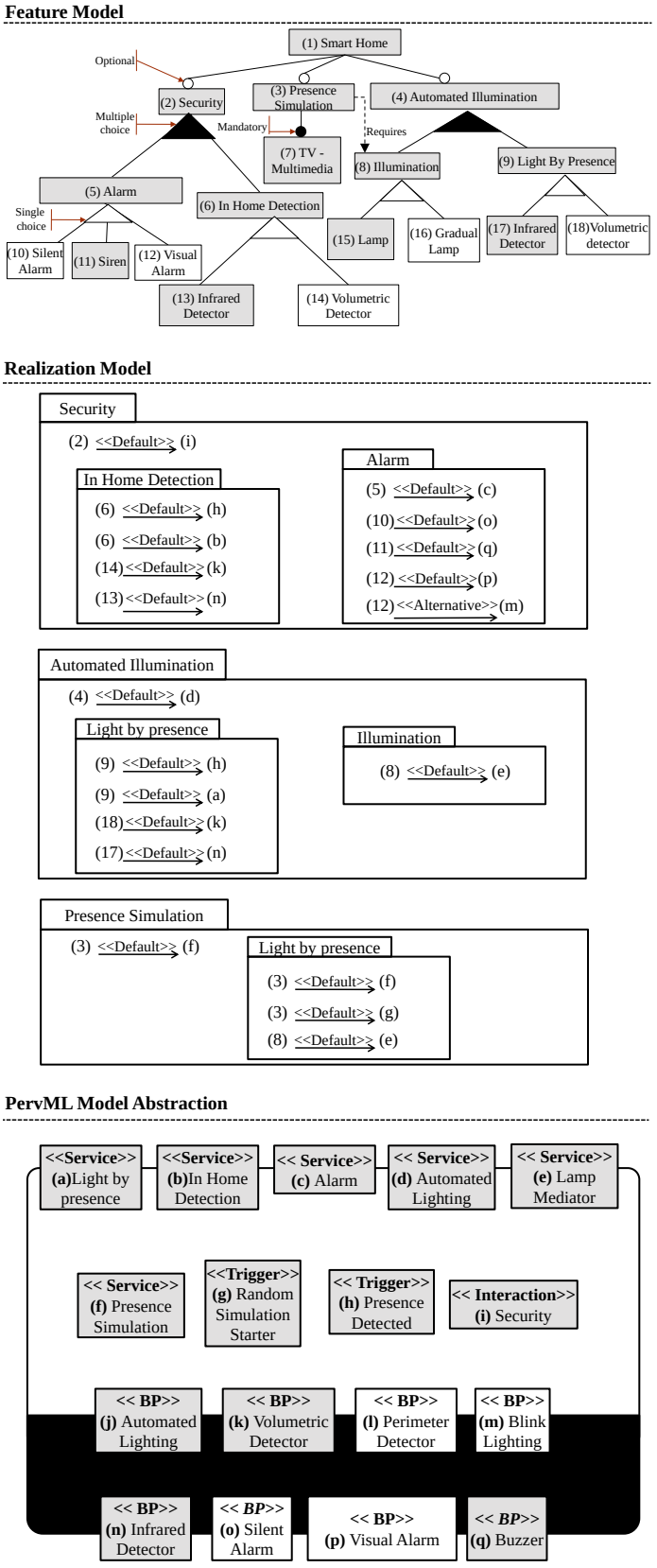

Fig. 3. Models for the SPL

D. Model To Text (M2T)

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 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.

At [1] there is more information about the transformation rules and the tools to support the code generation.

IV. Introducing end-users in the MDD-SPL

In the presented MDD-SPL, variability engineers set the smart home configuration by means of the feature model. Variability engineers make assumptions about the desirable functionality of end-users. Conversely, end-users are the ones who best know their activities and their functionality expectations. End-users and professional developers actually possess distinct types of knowledge. End-users are the "owners" of the problem and developers are the "owners" of the technology to solve the problem. End-users do not understand software developers' jargon and developers often do not understand end-users' jargon [29]. Although, end-users are not professional developers they have deep knowledge of their specific environment and they should be able to develop their own smart home system according to their needs. Hence, we involve end-users in the Smart Home configuration in order to minimize the mismatch between user expectations and system behaviour.

In order to tackle this, end-users must be supplied with visual development tools that allow them to describe their needs [30]. In this work, we have developed a tool that allows end-users to configure their smart home system using the MDD-SPL for smart homes described in the previous section. Fig. 4 shows an overview of the MDD-SPL with end-users. The end-user tool allows end-users to indicate which services and devices must be available in each location and configuring the feature model accordingly. Thus, when end-users have finished describing their needs, we obtain the decision model that determines the output system to be obtained by applying the model transformation.

To design the end-user front-end, we have based on well-accepted techniques and metaphors in the field of end-user development such as: Natural Programming, Programming By Example, Visual Programming and metaphors (see Section II). We have also applied interaction patterns and design principles to end-user interface design according to studies [5], [6], [7] which show how these patterns and principles help end-users (who may not have any background about computer applications). According to these studies, the main design interface decisions that we have applied are:

- **Using a wizard:** in our process the end-user needs to achieve a single goal (the description of their needed system) but several decisions need to be made before the goal can be fully achieved (several steps), which may not be known to the user. Thus, the use of a wizard is recommended in [5] since the user wants to reach the overall goal but may not be familiar with or interested in the steps that need to be performed.
- **Offering navigation buttons:** we use navigation buttons to suggest end-users that they are navigating a path with steps. This is recommended in [5] because the learning and memorization of the task of each step are improved. In addition, when users are forced to follow the order of tasks, they are less likely to miss important things and therefore will make fewer errors.
- **Displaying the elements using a grid layout:** this is recommended in [5] to any circumstance where several information objects are presented and arranged spatially within a limited area. This improves the presentation and it minimizes the time to scan, read and view objects on screen.
- **Offering options:** an interesting conclusion is reached in [6]: *what people see is what they select from!*. The study states that people tend to select from the entire list of options what they are first presented with. Rarely is an effort made to find additional options through scrolling. If eleven items are presented, the choice is from these eleven. When options must be compared among themselves, controls presenting all the options together will yield the best results.
- **Selection rather than introduce text:** the studies presented in [7] show the advantages and disadvantages of using either entry fields or selection fields for data collection. Since information became less familiar or subject to spelling or typing errors they recommend choosing a selection technique.

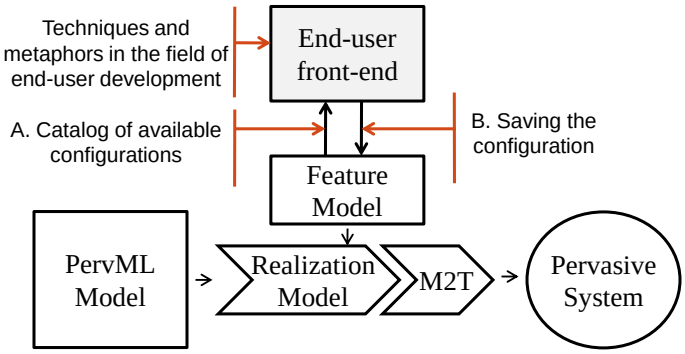

Fig. 4. Approach overview

Thus, we have developed a user interface based on the interface decisions presented above which allows end-users to specify the services and devices that they need. Fig. 5 shows a snapshot of this interface as end-users configure devices and services in their home. Each interface is divided into four areas: (1) Title and navigation buttons, (2) Catalog of available configurations, (3) End-user environment and (4) Information

[1] www.pros.upv.es/labs/projects/pervml

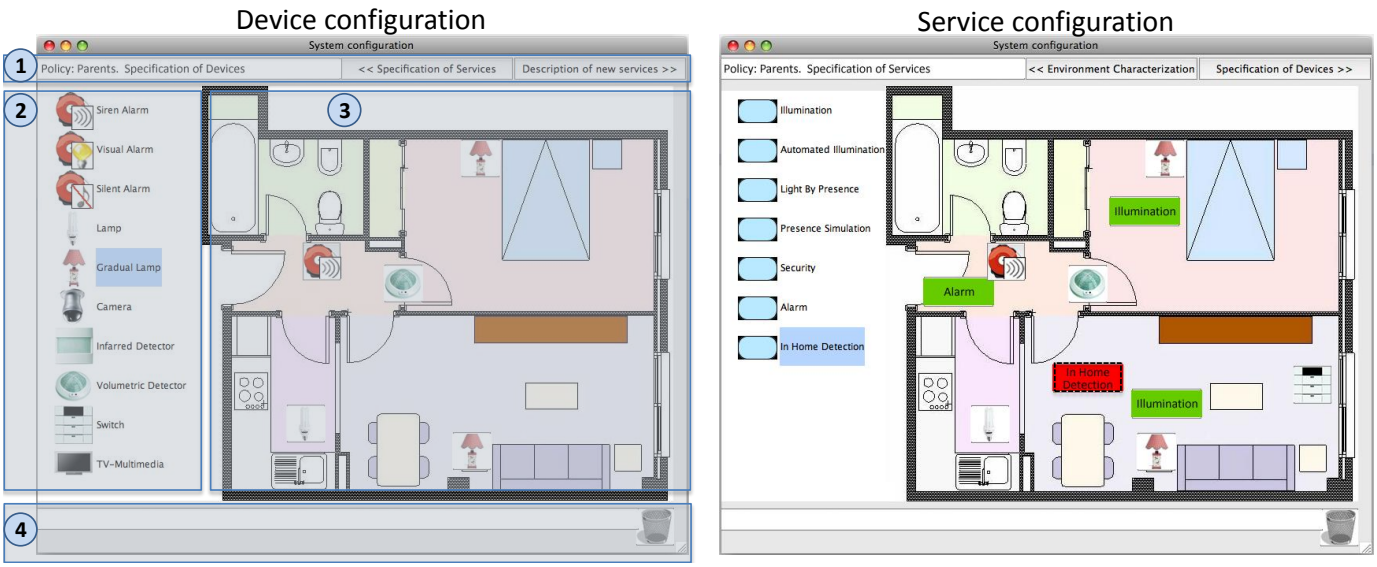

Fig. 5. Snapshot of the end-user front-end

where our tool can advise to end-users or assist them. In particular, we show at the left side of the figure how the end-user has selected some devices for different locations of their smart home (i.e. a siren alarm and a Volumetric detector for the corridor). At the right of the figure we show how the end-user has selected some services for different locations (i.e. Alarm service for the corridor).

As Fig. 5 shows, we have applied the interaction patterns described above. The *grid layout* pattern is applied to divide the interface into the four areas presented above. The *wizard pattern* is used to guide end-users along the process of creating a pervasive description by progressively asking them for the required information (services, devices, etc.). In addition, *navigation buttons* are also used in the area (1) to allow end-users to navigate between the different windows that ask for the required information. The *offer options* and *selection rather than introduce text* patterns are applied in the area (2) offering the devices/services available as options and allowing end-users to select these devices/services into the end-user environment represented in the area (3).

The next two subsections describe how the tool uses the Feature Model. Subsection A. describes how the end-user front-end uses the feature model to show the catalog of available configurations (see Fig. 4) and Subsection B. describes how the tool saves the configuration in the feature model activating/inactivating features according to the end-user's configurations.

A. Catalog of available configurations

As we described in subsection III-B, we use the feature model to describe the system configuration and its variants in terms of features. In the smart home domain, the system configuration that end-users have to select is made up of the services and devices required for each location in the environment.

At the top of Fig. 3 is shown the feature model which determines the initial and potential features of the smart home. These features represent services and families of devices. The **families of devices** are the leaves of the feature model and the **services** are the nodes which are not leaves. We specify families of devices in the feature model rather than devices because there is a large diversity of devices which are continuously changing. For each family of devices we offer a catalog of compatible devices. For example, the *Volumetric Detection* device family has a catalog of compatible devices which contains a Volumetric 360 degree detector as well as a 160 degree one. Thus, when a new device is supported all we have to do is update the catalog of that family of devices rather than the feature model.

Our tool shows end-users the options from the available configurations according to the feature model. Fig. 6 shows an example of service and device options according to the feature model. Note that these device options match the node leaves of the feature model presented in the figure (Siren Alarm, Visual Alarm, Siren Alarm, Infrared Detector and Volumetric Detector) and the service options match the nodes which are not leaves of the feature model (Security, Alarm and In Home Detection).

The available options are displayed in a tree. Studies described in [5] recommend using a tree when the number of groups is high. They also recommend that each option be explained so that users know of the consequences. Thus, we show in our tool a representative image for each service or device and a brief description. Fig. 5 shows the list of available devices and services that is shown to end-users from the feature model presented at the top of Fig. 3.

B. Saving the configuration in the feature model

Once the catalog of configurations has been shown, end-users can select services or devices for each location in

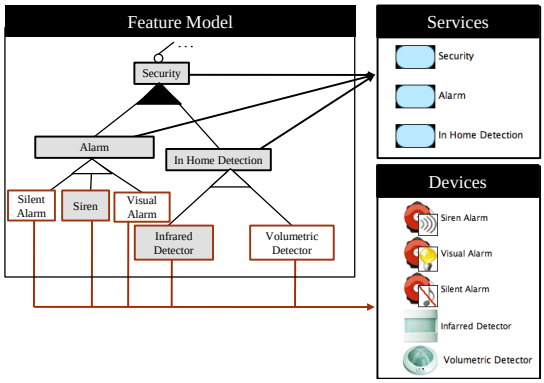

Fig. 6. The catalog of available configurations

the end-user's tool. When this happens, the end-user tool sets activated/inactivated features to the feature model. Thus, when end-users finish setting their system the feature model will have activated the features according to the end-user configuration.

To set a configuration end-users have to select the desired services and devices from the catalog of configurations and put them into the proper location. Then, a representative image of the service/device is displayed in the environment. The service/device can be displayed in two different colours: (1) red with a dotted frame if the service configuration has not fulfilled their constraints (services/devices that the service need) or (2) green if the service configuration has fulfilled their constraints. Fig.5 shows the specification of devices (see at the left side of the figure) and services (see at the right side of the figure).

In order to define these end-user interfaces we have based on the following end-user principles and interaction patterns [7], [5]:

- **Using autocompletion:** The study showed in [7] states that aided entry, also known as autocompletion, is preferred over unaided entry methods, and it is also the fastest method. Autocompletion reduces errors in comparison to unaided entry. In addition, it also minimizes the user's effort by reducing input time and keystrokes.
- **Using a warning:** this is recommended in situations where the user performs an action that may unintentionally lead to a problem [5] and the system cannot or should not automatically resolve this situation so the user needs to be consulted. The warning might also include a more detailed description of the situation to help the user make the appropriate decision by means of two options at least.
- **Offering all options:** this is recommended when the number of options is not large and they can be displayed without scrolling [7]. Rarely was an effort made to find additional options through scrolling.
- **Offering some options:** this is recommended when the number of options is high and it needs a scroll to be displayed. Thus, it is recommended to show some options of the available list [7]. This improves the speed of performance and satisfaction

According to the interaction patterns presented above, we have defined a set of mappings between the feature model and our end-user front-end and how the interaction patterns are used depending on the information that is available at the feature model. Next we present the interaction patterns used for each relationship of the feature model:

- If there is a **Mandatory feature**, we use **Autocompletion**. When a feature A is related to another feature B with a mandatory relationship, if A is selected B has to be selected too. In the end-user front-end, features are represented by services/devices. Thus, when the end-user selects a service that represents a feature A with a mandatory relationship to a feature B, the service representing feature B is automatically added to the same location of service A. For instance (see Fig. 7) when the end-user selects the *Presence Simulation* service for the living room (1) the *TV-Multimedia* device is automatically added to the same location (2) because there is mandatory relationship between the *Presence Simulation feature and the TV-Multimedia* feature. In addition, the feature model is updated by activating both features (3).

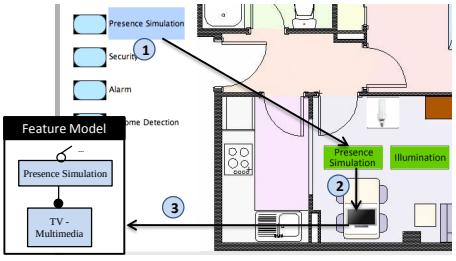

Fig. 7. Applying patterns in a mandatory relationship

- If there is a **Requires or Excludes feature**, we use **Warning**. When a feature A has a requires relationship with B, if A is selected feature B has to be selected too. Similarly, if feature A has an excludes relationship with B, when feature A is selected feature B does not have to be selected. In the end-user front-end, when the end-user selects a service that represents a feature with a requires or excludes relationship, the end-user front-end warns end-users by showing a warning. Fig. 8 shows when the end-user selects the *Presence Simulation* service for the living room (1). As the feature that represents this service has a requires relationship with the *Illumination* feature, the end-user front-end shows a Warning (2). Then the end-user adds this required service to the same location (3) and the feature model is updated activating the features *Illumination and Presence Simulation* (4).
- If there is an **Optional or single choice feature**, we use **Show all options and Autocompletion**. When a feature A has an *optional o single choice* relationship with other features, one of them has to be selected. In the end-user front-end, when the end-user selects a service

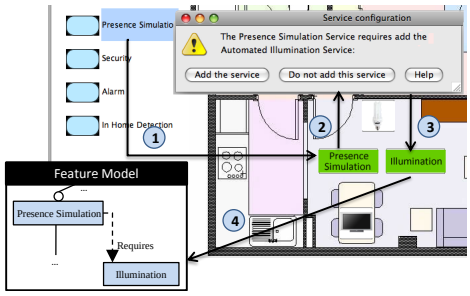

Fig. 8. Applying patterns in a requires relationship

that represents a feature with an optional or single choice relationship, the end-user front-end shows a dialog with all the services/devices that represent the related features. Thus, the end-user can select one of them and the end-user front-end adds the related service/device to the same location as the selected service/device. Finally, the feature model is updated. Fig 9 shows, as a representative example, how the end-user selects the *Alarm* service (1). This service represents a feature that has a Single Choice relationship. Then, a dialog is shown with all the devices that represent the related features (2). Then the end-user selects the *Siren* device and the feature model is updated activating the features *Alarm and Siren* (3).

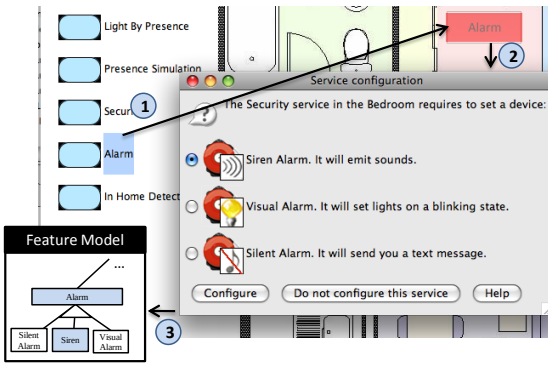

Fig. 9. Applying patterns in an optional or single choice relationship

- If there is a **Multiple choice**, we use **Show some options and autocompletion**. When a feature A has a multiple relationship with other features, one or more of them has to be selected. In the end-user front-end, when the end-user selects that represents a feature with a multiple relationship, the end-user front-end shows a dialog with services/devices that represents the related features. Then, the end-user can select one of more of them and the end-user front-end adds them to same location where the previously selected service is located. In addition, the feature model is updated according to this selection. Fig 10 shows, as representative example, how the end-user selects the *Security* service (1). The feature that represents this service has a *Multiple Choice* relationship. Then, a dialog is shown with the devices that represent the related features (2). Afterwards, the end-user selects the *Alarm* Service and the *In Home Detection* services. Finally, the Feature Model is updated activating the features *Security, Alarm, and In Home Detection* (3).

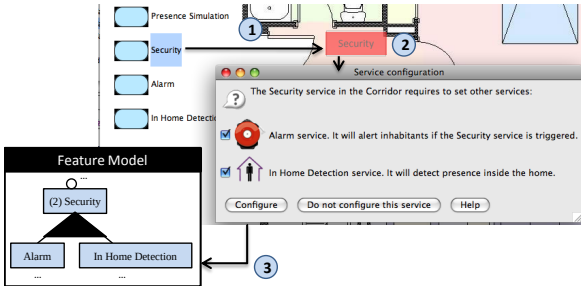

Fig. 10. Applying patterns in a multiple choice relationship

V. Supporting Technologies for the End-User Oriented MDD-SPL

As we described in the previous section, our end-user tool uses the Feature Model to offer the catalog of available services/devices. This model is also used to save the end-user's configurations by activating/inactivating features. The feature model is specified using the MOSkitt Feature Modeller editor [31], which uses the technology provided by the Eclipse Modelling Platform [32].

Thus, in order to connect the end-user front-end with the feature model we have used the EMF Model Query framework [33]. EMF Query provides an API to construct and execute query statements. These query statements can be used for discovering and modifying model elements. Queries are first constructed with their query clauses and then they are ready to be executed.

There are two query statements available: SELECT and UPDATE. The SELECT statement provides querying without modification while the UPDATE statement provides querying with modification. The SELECT statement requires two clauses, a "FROM" and a "WHERE." The FROM clause describes the source of model elements where SELECT can iterate in order to derive results. The WHERE clause describes the criteria for a model element that matches. The condition provided to the WHERE clause falls under a specialized condition called an EObjectCondition which is specially designed to evaluate model elements.

We have implemented the interaction patterns described in the Subsection IV-B by using EMF Model Query. For instance, when the end-user selects the Alarm service, the tool checks the feature model for the selected feature. It also checks the relations with other features. In this case, the Alarm service is related with a single choice relation with three features (Silent Alarm, Siren and Visual Alarm). Thus, as the feature model relation is Single choice, the interaction patterns that are applied are (see previous section): (1) Show all options and (2) Autocompletion. Then, we need both to obtain the features related with the selected one in order to show all of them, and to update the selected feature and also the selected related feature.

Next, we show the query that we have implemented to obtain the child features of the single choice relationship by using EMF Model Query:

```
SELECT statement =
 new SELECT(
  new FROM(currentFeature.getContents()),
  new WHERE(new EObjectReferenceValueCondition(
  new EObjectTypeRelationCondition(
        FeatureModelPackagePackage.eINSTANCE
            .getFeatureRelationship()),
  FeatureModelPackagePackage.eINSTANCE.
                getFeatureRelationship_From(),
  new EObjectInstanceCondition(SingleChoice))
  )
);
```

Given a feature (currentFeature) the select statement gets all the features related with the currentFeature with a single choice relationship (EObjectInstanceCondition(SingleChoice)). Then, these features are shown as service options on the dialog of Fig. 9.

Once the end-user chooses one of the presented options, the state of the selected feature and its related one is updated in the feature model from inactivated to activated. By contrast, if the end-user drops this kind of device into the trash, its state is updated to inactive.

VI. Conclusions and future work

Taking the advantage of current MDD techniques and an integrated SPL architecture, we have provided an interactive design tool that allows end-users (rather than engineers) to create tailored solutions that directly reflect their needs and expectations. In order to tackle this, we have presented an MDD-SPL approach based on Model Driven Development to develop smart home systems which is complemented with our end-user tool. We have also presented how the end-user tool gets and sets information of the feature model according to the end-user configurations. Furthermore, we have applied interaction patterns to the end-user tool which improve the user interface usability. Finally, we have presented the technology implementation for handling the feature model.

As future work, we plan to validate the end-user configurations in the end-user tool and assist end-users during the configuration process. To do this, we plan to use the feature model Analyser Framework [27]. Furthermore, we plan to involve end-users in the domain engineering phase. Our goal is the participation of end-users in the definition of new service configurations.

Acknowledgment

This work has been developed with the support of MEC under the project SESAMO TIN2007-62894 and cofinanced by FEDER, in the grants program FPI.

References

[1] M. K. Lee, S. Davidoff, J. Zimmerman, and A. K. Dey. Smart homes, families and control. In *Proceedings of Design & Emotion 2006*, 2006.

[2] Markus Voelter and Iris Groher. Product line implementation using aspect-oriented and model-driven software development. *SPLC 2007*, pages 233–242, Sept. 2007.

[3] Javier Muñoz and Vicente Pelechano. Building a software factory for pervasive systems development. In *CAiSE*, pages 342–356, 2005.

[4] C. Cetina, J. Fons, and V. Pelechano. Applying software product lines to build autonomic pervasive systems. pages 117–126, Sept. 2008.

[5] Martijn van Welie and Hallvard Trætteberg. Interaction patterns in user interfaces. In *PLoP 2000*, pages 13–16, 2000.

[6] Mick P. Couper, Roger Tourangeau, Frederick G. Conrad, and Scott D. Crawford. What they see is what we get: response options for web surveys. *Soc. Sci. Comput. Rev.*, 22(1):111–127, 2004.

[7] Wilbert O. Galitz. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[8] M. Anastasopoulos, T. Patzke, and M. Becker. Software product line technology for ambient intelligence applications. In *In Proc. Net.ObjectDays*, page 179Ű195, 2005.

[9] Jon O'Brien, Tom Rodden, Mark Rouncefield, and John Hughes. At home with the technology: an ethnographic study of a set-top-box trial. *ACM Trans. Comput.-Hum. Interact.*, 6(3):282–308, 1999.

[10] Rick Rabiser. Flexible and user-centered visualization support for product derivation. In *SPLC (2)*, pages 323–328, 2008.

[11] John Steinmetz. *Computers and Squeak as Environments for Learning*. 2000.

[12] David Canfield Smith, Allen Cypher, and Jim Spohrer. Kidsim: programming agents without a programming language. *Commun. ACM*, 37(7):54–67, 1994.

[13] Chin, Callaghan, and Clarke. An end-user programming paradigm for pervasive computing applications. *International Conference on Pervasive Services*, 0:325–328, 2006.

[14] Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. A cappella: programming by demonstration of context-aware applications. In *CHI '04*, pages 33–40, New York, USA, 2004.

[15] Jan Humble et al. Playing with the bits: User-configuration of ubiquitous domestic environments. In *UbiComp 2003*, 2003.

[16] Khai N. Truong, Elaine M. Huang, and Gregory D. Abowd. Camp: A magnetic poetry interface for end-user programming of capture applications for the home. In *Ubicomp 2004*, pages 143–160, 2004.

[17] Brad A. Myers, John F. Pane, and Andy Ko. Natural programming languages and environments. *Commun. ACM*, 47(9):47–52, September 2004.

[18] John Francis Pane. *A programming system for children that is designed for usability*. PhD thesis, Pittsburgh, PA, USA, 2002. Co-Chair-Myers,, Brad A. and Co-Chair-Garlan,, David.

[19] Henry Lieberman. Programming by example (introduction). *Commun. ACM*, 43(3):72–74, 2000.

[20] Andrew J. Ko and Brad A. Myers. Designing the whyline: a debugging interface for asking questions about program behavior. In *CHI '04*, pages 151–158, 2004.

[21] Carnegie Mellon University. Alice. http://www.alice.org/index.php, 1 2009.

[22] The accord toolkit. http://www.sics.se/accord/toolkit.html, 1 2009.

[23] Javier Muñoz and Vicente Pelechano. Applying software factories to pervasive systems: A platform specific framework. In *ICEIS (3)*, pages 337–342, 2006.

[24] Javier Muñoz, Vicente Pelechano, and Carlos Cetina. Implementing a pervasive meeting room: A model driven approach. In *IWUC*, pages 13–20, 2006.

[25] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. Generic semantics of feature diagrams. *Comput. Networks*, 51(2):456–479, 2007.

[26] D. Benavides, Ruiz A. Cortés, and P. Trinidad. Automated reasoning on feature models. *CAiSE 2005*, 3520:491–503, 2005.

[27] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A.Jimenez. Fama framework. In *SPLC*, 2008.

[28] Marcos Didonet Del Fabro, Jean Bézivin, and Patrick Valduriez. Weaving models with the eclipse amw plugin. In *Eclipse Modeling Symposium, Eclipse Summit Europe 2006, Esslingen, Germany*, 2006.

[29] Maria Francesca Costabile, Piero Mussio, Loredana Parasiliti Provenza, and Antonio Piccinno. End users as unwitting software developers. In *WEUSE '08*, pages 6–10, New York, USA, 2008.

[30] Henry Lieberman, Fabio Paternò, and Volker Wulf. *End User Development*. Springer, 2006.

[31] Moskitt feature modeller. www.pros.upv.es/labs/projects/mfm.

[32] Eclipse modelling framework. http://www.eclipse.org/modeling/.

[33] Emf model query. http://www.eclipse.org/modeling/emf/?project=query.