Mass Customisation along Lifecycle of Autonomic Homes*

Carlos Cetina[1] Pablo Trinidad[2] Vicente Pelechano[1]
Antonio Ruiz Cortés[2]
(1) Dpto. de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, E-46022, Spain
{ccetina, pele} at dsic dot upv.es
(2) Dpto. Lenguajes y Sistemas Informtticos
Universidad de Sevilla, Spain
{ptrinidad, aruiz} at us dot es

Abstract

Autonomic homes adapt themselves to give the user the best possible experience of the services they provide. They dynamically adapt its behavior at run-time in response to changing conditions in end-user needs and the surrounding environment devices. From the development point of view, producing and maintaining a large amount of autonomic homes need an affordable solution such as dynamic software product lines (DSPL). DSPL produce a set of products that share features and have the ability of reconfiguring at runtime. Since users maintain and modify their preferences in opportunistic and improvisational ways, an autonomic home must evolve in time according to user expectations. Current DSPL architectures implement the ability of reconfiguring a product but ignore user preferences. We present an extension to our DSPL architecture to incorporate user preferences so user customisation of autonomic homes is not limited to installation time but all along the lifetime.

1 Introduction

In the past few decades, many research efforts have been invested in developing the idea behind pervasive computing [10]. These efforts intend to incorporate interactive technology into our everyday environment, transforming the homes we live in into pervasive computing environments, which have been named as *Smart Homes*.

Some studies have highlighted that people continually reconfigure Smart Home spaces in order to meet their particular demands [8]. Additionally, smart home devices continually evolve as users upgrade them and install new devices to enrich the services a smart home offers. Users transform smart homes from conventional to personal. Therefore smart homes cannot be static any longer, they need to evolve as user preferences do.

Autonomic Computing [7] (AC) presents an alternative to support the evolution of systems without the need for human intervention. A system with autonomic capabilities is in charge of installing, configuring, tuning, and maintaining its own components at runtime. While the number of devices in our surroundings increases in smart homes, AC may help on reducing the configuration effort, simplifying user interaction with the system.

Autonomic homes are not only customised when they are produced to be delivered to final users, but also along their lifecycle due to extensions, failures or changes in user preferences. Producing and maintaining a huge amount of autonomic homes in their different states needs an affordable solution. A set of autonomic homes may share a high amount of devices and services, basically differing in some of them and how they are configured [4]. In the world of software engineering, Software Product Lines (SPL) propose a set of methods for the mass customisation of software products. SPLs are able to produce different products reusing common assets and allowing the customer to decide among a bunch of optional features. A Dynamic SPL [5] (DSPL) is a particular kind of SPL that produces products which can reconfigure themselves at runtime. In our previous work [3] we successfully built a DSPL that produced customised Autonomic Homes. Two main advantages arise from using DSPLs to build a family of autonomic homes:

1. DSPLs are suitable for economies of scope as they

*This work was partially supported by the European Commission (FEDER) and Spanish Government under Web-Factories (TIN2006-00472) and SESAMO (TIN2007-62894) projects and by the Andalusian Government under project ISABEL (TIC-2533).

maximise software reuse analysing the commonality among products from scratch rather than when each product is built.

2. DSPLs produce systems capable of adapting themselves to fluctuations in user preferences and environment and evolving devices constraints.

Although technology weaves our everyday life, people is reluctant to technology to make autonomic decisions without supervision. People need to feel that they control technology instead of technology controls them. Emergent behaviours in autonomic homes impede users from understanding system reactions, and also reduce the predictability of the system. Users need to feel that although an autonomic home makes its own decisions, their preferences are still taken into account. We show how DSPLs are suitable for the mass customisation of autonomic homes that automatically change their configuration according to the current situation with minimal user intervention while user preferences are still taken into account. To perform this reconfiguration, our approach focuses on covering the average demand of the home inhabitants rather than the preferences of specific individuals.

2 Autonomic Computing and DSPL for Autonomic Homes

Users may find smart homes function overly complex, if not unnecessary, and prefer a simplified control of their systems [6]. Autonomic Homes intend to reduce human intervention to minimal. The following autonomic computing characteristics play a key role in achieving an unobtrusive evolution of Smart Home environments:

Self-configuring. New kinds of devices may be incorporated to the system. For example, when a new presence sensor is added at a home location, the different smart home services such as security or lighting control should automatically make use of it needing no configuration action from the user.

Self-healing. Whenever a device is removed or fails, the system should adapt itself to offer its services in an alternative way in order to reduce the impact of the device loss. For example, if an alarm fails, the Smart Home may blink the lights as a replacement for the failed alarm.

Self-adapting. The system should adjust its services in order to fulfill user needs and environmental changes accordingly. For example, when a user leaves home, services in the home should be reorganized to give priority to security.

Many approaches incorporate AC into Smart Homes relying on different techniques such as stochastic models, reinforcement learning and control theory. However, as emergent behaviours rise, they impede users from understanding system reactions, and also reduce the predictability of the system. Therefore, any simplification in user interaction implies a reduction in system predictability. Our purpose is breaking with this implication, proposing a solution that simplifies user interaction while prediction is still met taking into account user preferences.

In our previous work, AC characteristics are provided by DSPL architectures. The benefits of using DSPL for our purpose is twofold: 1) We get a reference architecture to perform reconfiguration at runtime and 2) We are able to produce a set of products instead of only one, setting up a SPL of AC systems. The core elements in a DSPL architecture are components (services and devices) and communication channels among them. The reconfiguration is dynamically produced changing communication channels and adding or removing components. Figure 1 uses PervML[1], a domain-specific language for pervasive systems, to show a scenario where a smart home is reconfigured as a user leaves. Such home has several devices (TV, Lights, presence sensors and an alarm) and services that use them (multimedia, lighting and security services and a presence simulator). In an initial state, lights turn on and off depending on user presence and multimedia service only works on user demand. When user leaves and there is nobody at home, presence simulator service activates and some of the communication channels change to turn on and off lights and TV to simulate presence. Security service uses presence sensors to raise the alarm if any presence is detected.

Self-adapting a system is not simple since the autonomic home must compose a suitable reconfiguration plan to perform the changes in the architecture. Not every configuration is allowed and not only user preferences must be taken into account but also component constraints. To represent what may change and what may not in an autonomic home we use feature models, a model that is commonly used to describe the variability in SPLs. A feature model describes the products a SPL is able to build in terms of its available features, i.e. an increment in product functionality. Features are hierarchically linked in a tree-like structure through variability relationships such as optional, mandatory, single and multiple choice. A product built by a DSPL contains a set of active and deactivated features that must satisfy the constraints imposed by the relationships in the feature model. Therefore constraints limit the potential combinations of features and must be taken into account prior any change in a product configuration.

Since features represent coarse-grained functionality, we remark that the mapping between features and architectural components is not straightforward. A feature can be supported by many architectural components, and each component may support more than one feature. For example, a single presence sensor can be used to support different fea-

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

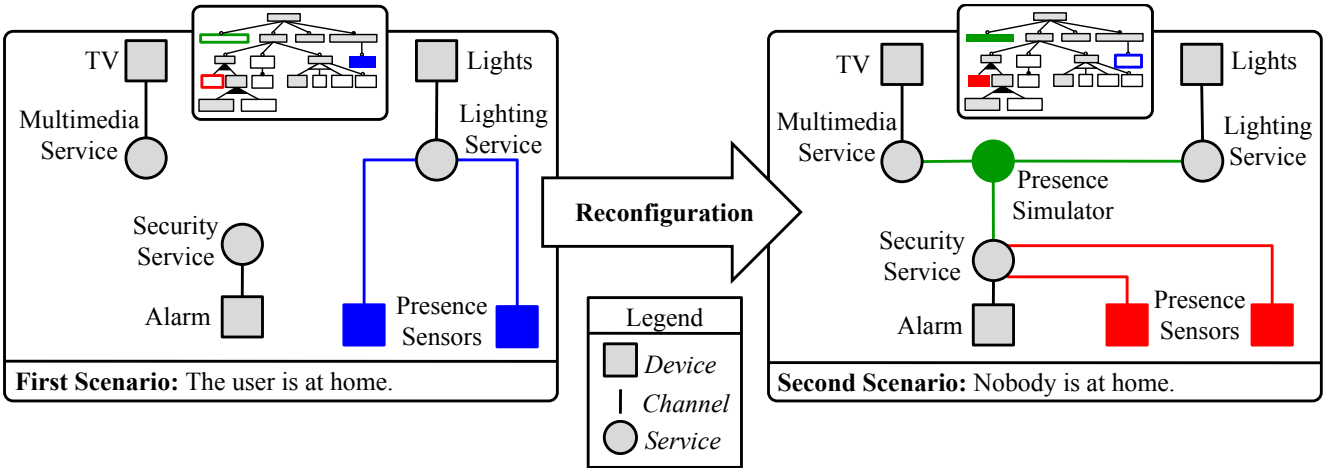

Figure 1. Impact of active features on system components for two scenarios.

tures such as security (detect intruders) and lighting control (turn lights on when someone approaches).

The feature model in Figure 2 describes an autonomic home with automated lighting, multimedia and security. At any time, some features are active (grey-coloured) while others are deactive (white-coloured). As external conditions change system reacts activating or deactivating features taking into account the constraints in the feature model. We refer to *resolutions* as a list of features to be activated or deactivated to deal with a change in the context.

A resolution must take into account three factors: available components, component constraints and user preferences. Our previous work only dealt with first two of them, leaving user preferences at a secondary level. In this article we show how we have extended our previous solution to support multiple user preferences in system reconfigurations.

3 Multiple Users in Autonomic Homes

Our first approach to build autonomic homes using DSPLs fully supported self-configuring and self-healing properties as any autonomic home was able to reconfigure itself on devices failure, removal or addition. However, self-adapting was limited as user preferences were considered in the order they were produced, giving priority to those preferences that came last. It produces situations where users contradict themselves continuously, ignoring any kind of prioritisation. For example, kids should not contradict their parents decisions and there is no way parental control for TV contents could be removed; if there are 4 adults watching TV in a sitting room and lights are turned off, no other user may turn on lights since the preferences of 4 people is more relevant than one person decision. However, if there are 4 kids watching TV and an adult turns on the lights, maybe the decision must be respected. Of course if a flooding occurs while anyone is watching TV, lights turn on and alarm sounds independently from any user preference. Notice that external providers have the consideration of user at all effects, expressing their preferences regarding other features of the system.

As it may be observed, three issues must be considered to deal with multiple users:

1. There exist different kinds of user categories that vary on the home.
2. There may be multiple users each one belonging to a user category at home at the same time, each one with his/her preferences.
3. Users may change their preferences at any given moment.

Whenever new users appear at home, leave or their preferences change, an autonomic home must analyse its state and determine if there exist another configuration that satisfies most of the user preferences. The objective of self-adapting an autonomic home is reconfiguring its architecture to maximise the fulfilment of user preferences. Users express their preferences selecting the features they want to be activate, deactivated or they have no preference for. Sometimes, every user preferences may be fulfilled at the same time; other times some user preferences could not be partially or completely satisfied as they collide with other user preferences. To break a deadlock, user categories are assigned a weight that is used together with the number of users preferring a feature activation or deactivation, stating some kind of democratic system.

But not only preferences are relevant for self-adapting, but also the constraints imposed by the feature model which represents the dependencies and incompatibilities among

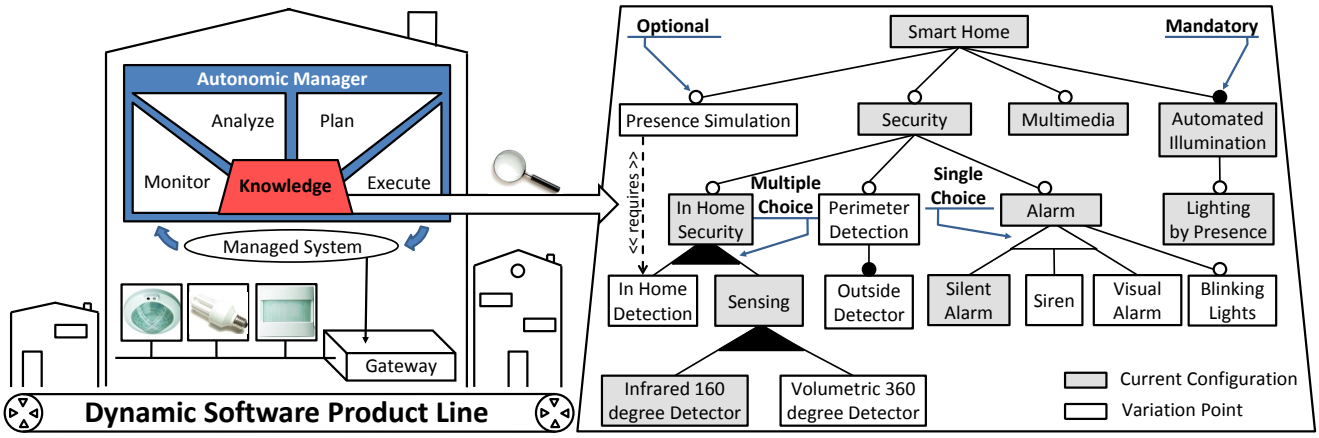

Figure 2. a Feature Model and a feature selection defines a specific home within a family of Autonomic Homes.

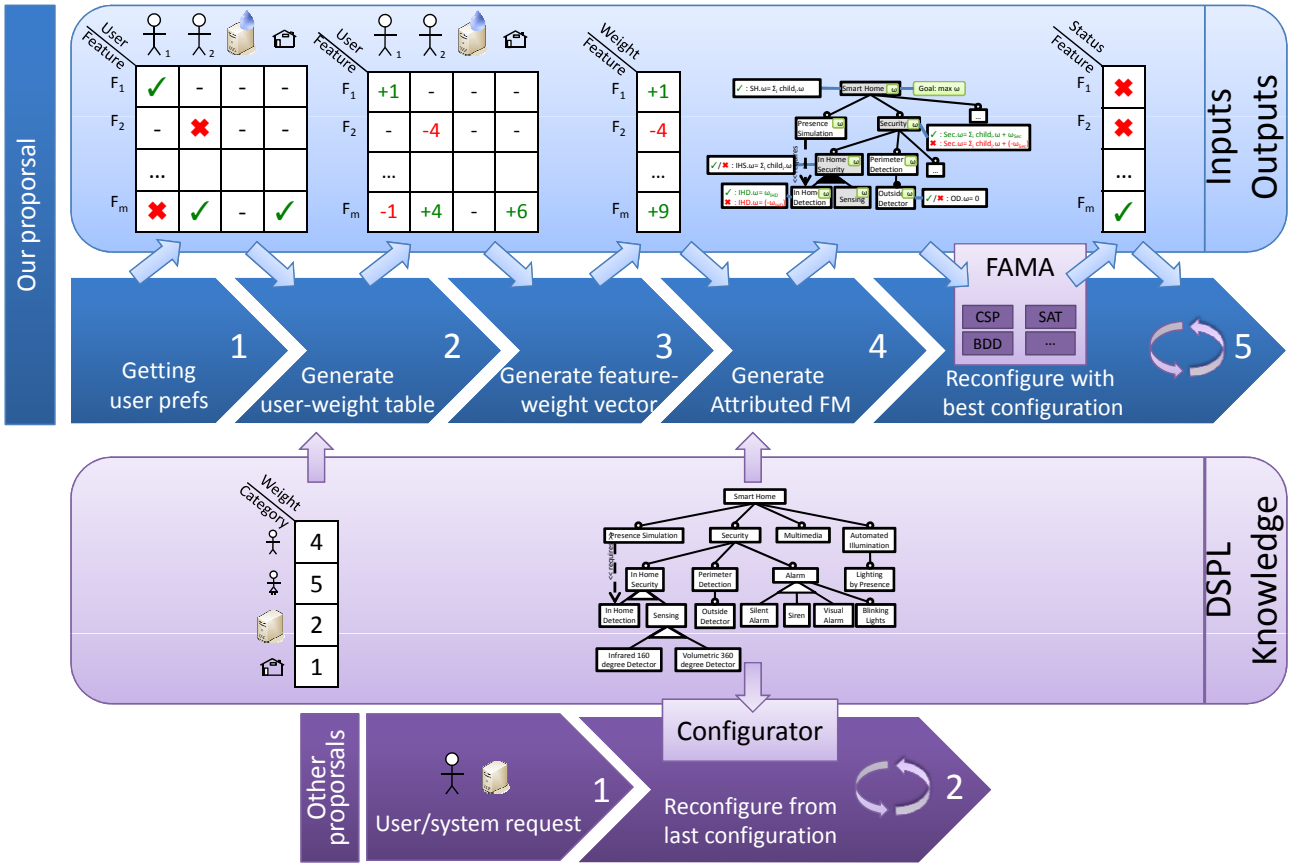

Figure 3. Reconfiguration process from user preferences and feature model constraints

features and even resource limits. Next, we describe the process to obtain a reconfiguration taking into account user preferences and feature constraints. We divide the process in 5 steps which are depicted in Figure 3 and detailed below.

Step 1: getting user preferences

Users and external services express their preferences as the subset features they want to be de/activated while remaining features are unrelevant for them. These preferences are represented in a table of preferences per feature. If a user leaves home or a service becomes unavailable, it is removed from the table so the table only contains preferences of the users that are at home and available services.

Step 2: generating user weights table

Every user belongs to a category which is assigned a natural number that represents the importance or weight of the preferences of its users. These weights are used to transform a preferences table into a table of weights where each feature selection is assigned the weight from the category a user belongs to. A feature activation in preferences table is substituted by the weight of its corresponding user; a feature deactivation is represented by the negation of the weight. If a user expresses no preference about a feature, no weight is assigned and its value will be ignored in following steps. Notice that $+4$ weight represents a higher trend to feature activation than $+1$ and -4 indicates a higher trend to feature deactivation than -1.

To mark a feature that must be de/activated due to an emergency independently from user preferences, we set its weight to $+\infty$ for activation or $-\infty$ for deactivation.

Step 3: generating weights-per-feature vector

The objective of this step is obtaining an average weight for each feature that gives an idea of the importance the users give to its de/activation. Weights table is projected into a weights vector where each feature is assigned an average weight obtained from a projection function. In our case, projection function just sums up all the weights for a feature although other criteria may be considered. A positive average weight for a feature tends the system towards such a feature activation; a negative weight suggests a feature deactivation; no weight or a null weight points out an indifference of a feature de/activation.

Step 4: generating an attributed feature model

The weights in the vector define the configuration that maximises user preferences. However, the configuration must be validated if it satisfies all the constraints in the feature model. To achieve it, existing feature model analysis tools such as FAMA Framework [9] are used. In case the configuration does not satisfy every user needs, we are interested in just a good configuration that satisfies most of them. Expressing this problem in terms of feature weights, we search for a configuration that weights the most.

Besides expressing contraints, feature models may be attributed[1] with additional information and perform analysis operations over them. Every feature within the feature model is enriched with weights allowing to perform analysis operations on them, such as finding the configuration that maximises the weight of the overall product.

An attribution of feature model with weights is partially shown in Figure 4 where a weight attribute (ω) is added for each attribute. Satisfying user preferences for a feature sums up its weight to its parent feature. Ignoring them substracts its weight to its parent feature. Following this hierarchical relation, the overall weight of a configuration is finally obtained for the root feature.

As infinite values would affect the overall result, the affected features are directly de/selected and their weight just sums up child features weights, as shown for *In Home Security* feature. Null weights are interpreted as zero to avoid interferring in weights calculation.

Step 5: Reconfiguring home with the best configuration

Once the feature model is attributed, obtaining a configuration that satisfies users consists of searching for the one whose weight is the highest possible considering feature model constraints. Such optimization operation is performed by FAMA Framework [9]. The tool returns a valid and best configuration for the attributed feature model that is used to reconfigure the home.

3.1 Special cases

The reconfiguration process offers a framework that may support some exceptional cases such as follows:

1. Exceptions: weights assigned to categories and therefore to users set a hierarchical structure that is not as flexible as needed for some situations. For example, if an alarm raises no user is allowed to turn it off and reconfigure the home but the security company. The security company has a higher weight than any other user for security features, but it makes no sense that they have any weight to change multimedia configuration. In case category weights are not sufficient to describe an scenario, step 1 may be ignored and user weights table is directly filled considering any exception.

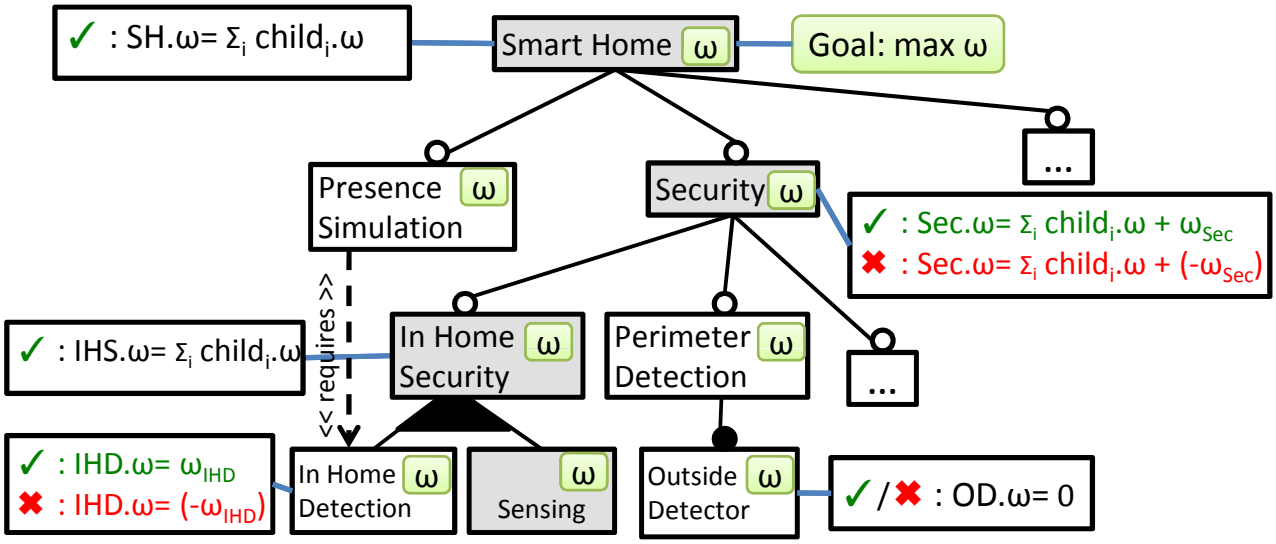

Figure 4. A Feature Model attributed with weights

2. Democracy: an autonomic home where every category is assigned the same weight implements a democratic scenario where decisions are taken relying on the number of users rather than their weights. Meetings or parties are situations where democracy implements a nice solution.

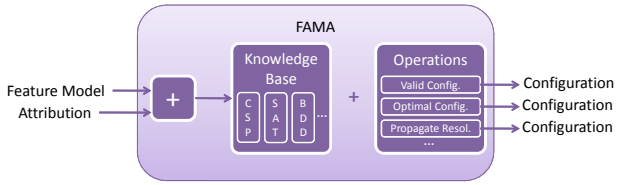

Figure 5. FAMA working

4 Let's make it real

Above steps present quite simple algorithms to perform the autonomic capabilities in autonomic homes. Step 5 is the most complex one and is delegated to FAMA Framework (FW) [2]. FAMA FW is an open-source analysis tool that uses constraint solvers, boolean satisfiability problems, binary decision diagrams and many other solvers to solve analysis operations on feature models. FAMA FW chooses the most suitable solver for each requested operation searching for the fastest response time. It supports many different operations on attributed feature model being attribute optimization one of them.

Finding an optimal configuration is a hard problem and may take time to be solved in some situations. It is important to give the fastest response as possible. But fast is frequently incompatible with best solution. It is possible to limit search in time so not the best but a good configuration is obtained. It is still possible to keep the tool searching for a better or the best solution in background so later reconfigurations may arise whenever they are found.

[2] http://www.isa.us.es/fama

4.1 Evaluation of the Proposal

We have implemented a prototype to evaluate the feasibility of this approach in real life deployment sessions. In the experimental set-up, a scale environment with real devices is used to represent an Autonomic Home. In this way, a researcher may interact with the same devices that can be found in a real deployment as a user would do (see top-left side of Fig. 6). A visualization tool is used during the experiments in order to keep track of the system evolution. This tool graphically depicts the resources, services and connections that are available in the system at any given moment (see bottom-left side of Fig. 6). Since an autonomic behaviour is usually a response to context and environmental conditions, some mechanisms are provided for triggering them. We have adopted RFID cards to set the smart home context (see right side of Fig. 6). Each of the cards symbolizes users or events in the Smart Home. These cards are combined to generate events and to trigger reconfigurations in the Smart Home following the procedure presented in previous section.

As an example of scenario simulated with the experi-

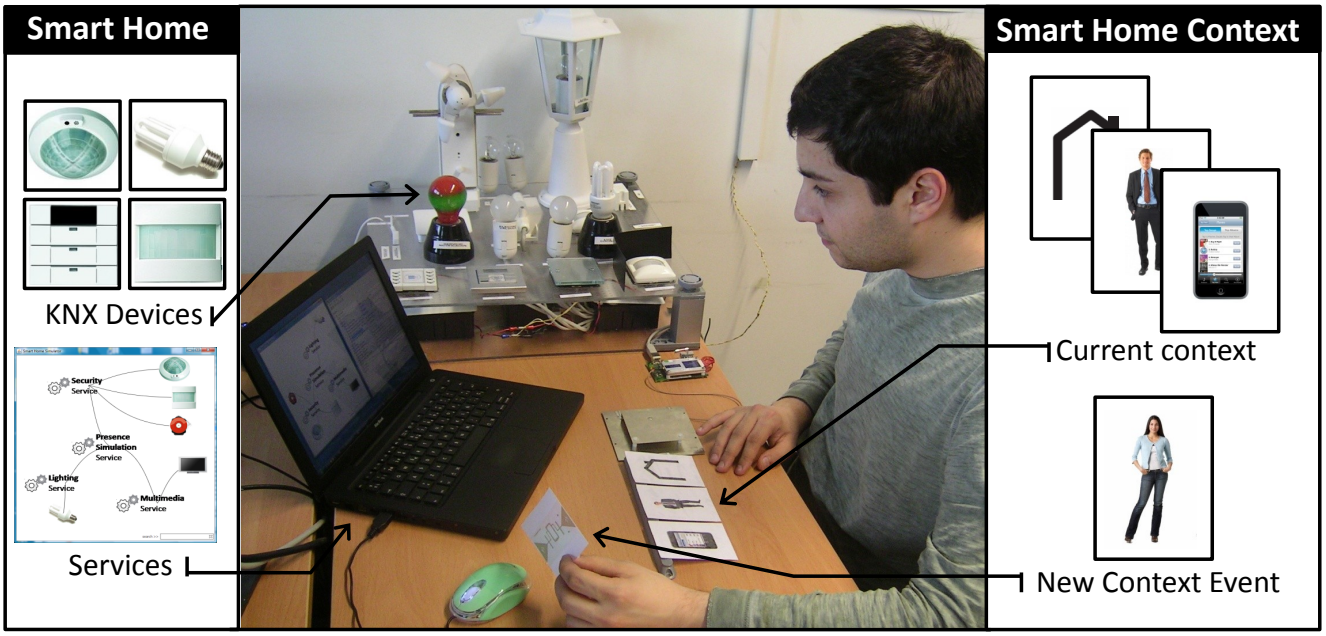

Figure 6. experimentation set-up.

ment, activating a presence detector produces different results according to the current context expressed by the RFID cards. For example, an initial scenario could consist in a Smart Home where one inhabitant is at home as illustrated in Fig. 6. The system architecture is organized in such a way that the piped music is available and presence sensors are used by the lighting service. The researcher may listen to the music and the lights are turned on/off as the researcher interacts with the sensors. The prototype is configured to give priority to comfort.

If the researcher removes the card that represents the home inhabitant, a reconfiguration is triggered and the sensors are automatically no longer used for the purpose of light control but for security instead. As a consequence, when the researcher interacts with the sensors again, the alarm raises. The prototype is configured to give priority to security.

The above scenario is part of our catalog to test the autonomic capabilities of our architecture in different adaptation scenarios. More details about these scenarios can be found in [2] and at http://www.autonomic-homes.com where several videos and screen casts show our prototype in action.

Creating an Autonomic Home is not an interesting problem if the result does not suit user expectations. The development of sophisticated but users-accepted Autonomic Homes is the exciting and challenging problem. By combining DSPLs and autonomic computing, our intent is to focus on commonalities and abstractions that are valid across a set of users, looking for a trade-off between Personalization and Reusability. That is, our approach focuses on covering the average demand of the home inhabitants rather than the preferences of specific individuals only.

References

[1] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, 3520:491–503, 2005.

[2] C. Cetina, J. Fons, and V. Pelechano. Applying Software Product Lines to Build Autonomic Pervasive Systems. *Software Product Line Conference, 2008. SPLC 2008. 12th International*, 8-12 Sept. 2008.

[3] C. Cetina, P. Trinidad, V. . Pelechano, and A. Ruiz-Cortés. An architectural discussion on dspl. *2nd International Workshop on Dynamic Software Product Line (DSPL08)*, 2008.

[4] J. Coplien, D. Hoffman, and D. Weiss. Commonality and variability in software engineering. *Software, IEEE*, 15(6):37–45, Nov/Dec 1998.

[5] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid. Dynamic software product lines. *IEEE Computer*, 41(4):93–95, 2008.

[6] R. Harper. Inside the smart home. pages xi, 264 p. ;, 2003.

[7] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[8] J. O'Brien, T. Rodden, M. Rouncefield, and J. 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.

[9] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A.Jimenez. Fama framework. In *12th Software Product Lines Conference (SPLC)*, 2008.

[10] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, September 1991.