Prototyping Dynamic Software Product Lines to Evaluate Run-Time Reconfigurations

Carlos Cetina, Pau Giner, Joan Fons, Vicente Pelechano

Centro de Investigación en Métodos de Producción de Software
Universidad Politécnica de Valencia
Camino de Vera s/n, E-46022, Spain
{*ccetina, pginer, jjfons, pele*}*@dsic.upv.es*

Abstract

Dynamic Software Product Lines (DSPL) encompass systems that are capable of modifying their own behavior with respect to changes in their operating environment by using run-time reconfigurations. A failure in these reconfigurations can directly impact the user experience since the reconfigurations are performed when the system is already under the users control. In this work, we prototype a Smart Hotel DSPL to evaluate the reliability-based risk of the DSPL reconfigurations, specifically, the probability of malfunctioning (Availability) and the consequences of malfunctioning (Severity). This DSPL prototype was performed with the participation of human subjects by means of a Smart Hotel case study which was deployed with real devices. Moreover, we successfully identified and addressed two challenges associated with the involvement of human subjects in DSPL prototyping: enabling participants to (1) trigger the run-time reconfigurations and to (2) understand the effects of the reconfigurations. The evaluation of the case study reveals positive results regarding both Availability and Severity. However, the participant feedback highlights issues with recovering from a failed reconfiguration or a reconfiguration triggered by mistake. To address these issues, we discuss some guidelines learned in the case study. Finally, although the results achieved by the DSPL may be considered satisfactory for its particular domain, DSPL engineers must provide users with more control over the reconfigurations or the users will not be comfortable with DSPLs.

Key words: Dynamic Software Product Line, Variability Modeling, Smart Hotel

1. Introduction

Variability modelling (in the context of Software Product Line engineering) has proven itself to be an efficient way for dealing with varying user needs and resource constraints. However, the focus has been on the efficient derivation of customized product variants that, once created, keep their properties throughout their lifetime. Previous research [24, 19, 3] shows that the combination of variability modelling and Dynamic Software Product Lines (DSPL) can assist a system to determine the steps that are necessary to reconfigure itself. Specifically, these systems can activate/deactivate their own features dynamically at run-time according to the fulfillment of context conditions. These features represent increments in the system functionality.

Variability models specify the possible configurations of a system, while a Dynamic Product Line Architecture can be rapidly retargeted to a specific configuration. That is, software variants are managed and variation points are bound flexibly, but all this is done at runtime and fully automatic. On the one hand, since the models that form the basis for run-time reconfiguration are available at design time, it is possible to validate reconfigurations at an early stage of the development process without first implementing them [5]. However, not all potential run-time failures can be anticipated during system design [26]. On the other hand, no reliable reconfigurations may desynchronize the model level and the architecture level. Since the reconfiguration is driven by the variability models and fully automated, this desynchronisation would lead the system to unexpected results in the next reconfigurations.

In this work, we are concerned with reliability-based risk of run-time reconfigurations. To this end, we have adopted the definition in [37], which defines reliability risk as a combination of two factors: the probability of malfunctioning (Availability) and the consequences of malfunctioning (Severity). We have considered these aspects since they are especially relevant when dealing with DSPLs. For traditional Software Product Lines, once a product is obtained for a given configuration, it can be tested intensively before it reaches the end-users. However, the case of DSPLs is different since different configurations are obtained at run-time. A failure in DSPL reconfigurations directly impacts the user experience since the reconfiguration is performed when the system is already under the user control. Thus, our hypothesis is that users require a controlled risk of run-time reconfigurations in order to feel comfortable with DSPLs.

To evaluate the availability and severity of the run-time reconfigurations, we have developed a Smart Hotel case study. The Smart Hotel reconfigures its services according to changes in the surrounding context. A hotel room changes its features depending on users' activities to make their stay as pleasant as possible. Overall, the case study comprises eight scenarios and eighteen reconfigurations among these scenarios. Each scenario denotes the specific configuration of the system in terms of active features and architecture components such as services, devices and communication channels.

The run-time reconfiguration among the different scenarios is the main unit of analysis that we address in this case study. This case study was deployed in a scale environment with real devices to represent the Smart Hotel with human subjects participating in the evaluation.

Two major challenges were identified and addressed with the involvement of human subjects in the evaluation. On the one hand, reconfigurations are triggered by context events many of which are difficult to be reproduced in practice (e.g., a fire). To address this challenge, we have developed a technique that is based on RFID-enabled cards to easily specify the current context. On the other hand, when reconfigurations are performed, some of the effects are easily perceived (e.g., an alarm is triggered) while others are not (e.g., some sensors are deactivated). Thus, we consider that the direct observation of the physical devices is not enough for evaluating the run-time reconfigurations. To address this challenge, we provided participants with a configuration viewer tool which helps them to understand and evaluate the effects of the reconfigurations.

In this paper, we present the Smart Hotel case study driven by the run-time reconfigurations of a DSPL. The case study was deployed with real devices and human subjects were involved to evaluate the reliability-based risk of reconfigurations. Specifically, we focus on two attributes of reliability-based risk: probability of malfunctioning (Availability) and the consequences of malfunctioning (Severity). Moreover, we successfully identified and addressed two challenges associated with the involvement of human subjects: enabling participants to (1) trigger the run-time reconfigurations, and to (2) understand the effects of the reconfigurations.

The evaluation of the DSPL reveals positive results regarding both Availability and Severity. However, participant feedback highlights issues with the recovery from a failed reconfiguration or a reconfiguration that is triggered by mistake. To address these issues, we discuss some guidelines learned in the case study. Finally, we conclude that the DSPL achieve satisfactory results with regard to reliability-based risk; nevertheless, DSPL engineers must provide users with more control over the reconfigurations or they will not be comfortable with DSPLs.

The paper is organized as follows. Section 2 presents the infrastructure of the DSPL prototype: Variability Modelling, DSPL architecture, and Context Modelling. Section 3 provides an overview of the Smart Hotel case study and also gives details about its reconfiguration scenarios. Section 4 describes the experimentation logistics and the challenges addressed. Section 5 introduces the results of the evaluation. Section 6 discusses the guidelines learned in the case study. Finally, Section 7 provides an overview of related work, and section 8 concludes the paper.

2. DSPL Prototype Overview

In the present work, the participants were involved in a case study to evaluate the run-time reconfigurations among the different scenarios of a Smart Hotel. Many approaches can be applied to realize the dynamic adaptation of the Smart Hotel such as stochastic model-based [2], reinforcement learning-based [33] or control theory-based [28]. However, this work focuses on domains where it does not seem economically realistic to meet the individual requirements of each potential user. 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* as DSPLs do.

The combination of (1) the variability analysis [7] from SPLs and (2) the capability of DSPL architectures to activate/deactivate their own features at run-time reduces production costs to the expense of limiting the level of detail in dynamic adaptation. This trade-off is acceptable in these domains, such as the Smart Hotel, since in general the focus is on covering the average demand, not the needs of each individual.

Next, we present a brief overview of Variability Modelling, DSPL Architectures, and Context Modelling, and how these techniques are combined to support the reconfigurations of the Smart Hotel case study as follows:

Variability Modelling: From the different techniques that are suited for variability analysis, we have chosen feature modelling [8] because it has good tool support for variability reasoning [1]. Feature modelling is widely used for the specification of system functionality in a coarse-grained fashion by means of the feature concept (an increment in system functionality). As illustrated in Fig. 1, the features are hierarchically

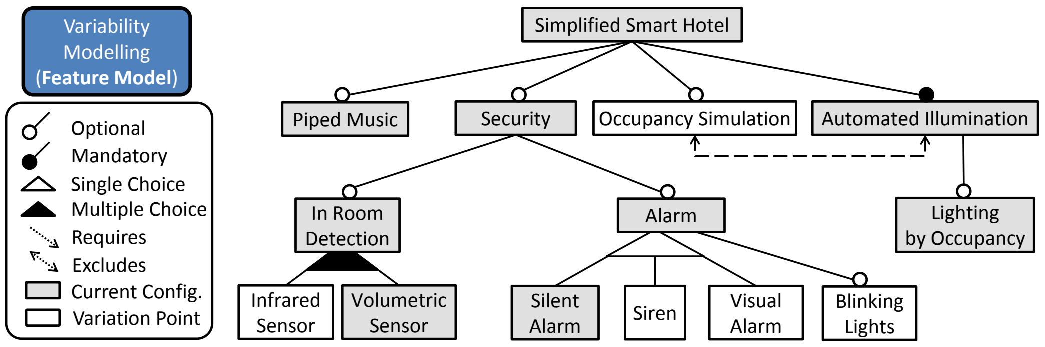


Figure 1: Feature model of a simplified Smart Hotel.

linked in a tree-like structure through variability relationships such as optional, mandatory, single choice, and multiple choice.

The Feature Model of Fig. 1 describes a simplified Smart Hotel with *Piped Music*, *Security*, and *Automated Illumination*. The grey features are the current features of the smart hotel, while the white features represent potential variants as they may be activated in the future.

We let $[\![FM]\!]$ denote the set of all Features (active or inactive) in a Feature Model. We define the Current Configuration (CC) of a system as the set of all active features (F) in a Feature model.

$$CC \overset{\text{def}}{=} \{F\} \mid F \in [\![FM]\!] \wedge F.state = \mathit{Active} \wedge CC \subseteq FM$$

For example, the CC of the Feature Model in Fig. 1 is expressed as follows:

$$\begin{aligned} CC_{Fig.1} = \{ & SimplifiedSmartHotel, PipedMusic, Security, \\ & InRoomDetection, Alarm, VolumetricSensor, \\ & SilentAlarm, \mathit{AutomatedIllumination}\} \end{aligned}$$

Dynamic Product Line Architecture: In order to provide a flexible reconfiguration, we have considered an architecture that is based on different components with communication channels. We classify these components into two categories: Services and Devices. Each Service coordinates the interaction between devices to accomplish specific tasks (these devices can be hardware o software entities).

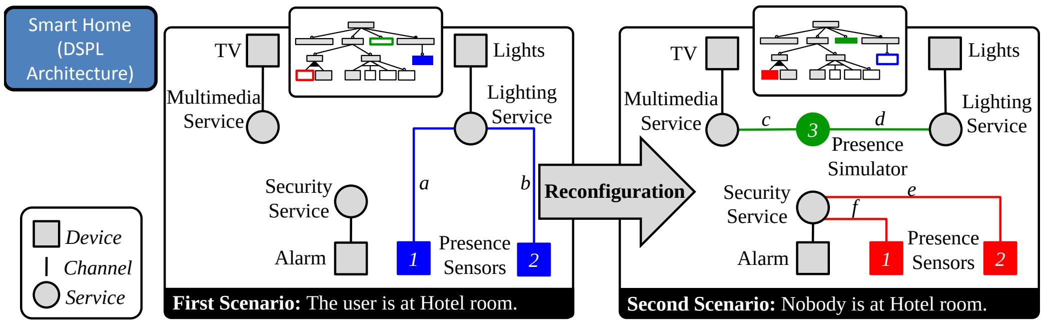


Figure 2: Impact of active features on system components for two scenarios.

This architecture allows an easy reconfiguration since communication channels can be established dynamically between the components, and these components can dynamically appear or disappear from configurations [19].

To achieve the dynamic reconfiguration of the architecture components, we apply software reconfiguration patterns [17]. Software reconfiguration patterns provide a solution to a reconfiguration problem where the configuration needs to be updated while the system is operational.

Specifically, we use the *Decentralized Control System Reconfiguration* [15]. In this pattern, components notify each other if going to a passive state. Notified components can cease the communication with its neighbor component (which is going to a passive state), but can continue with other component communications.

This pattern provides the following properties to the architecture: (1) Non interference with those parts of the application that are not impacted by the reconfiguration, and (2) during reconfiguration, impacted components must complete their current computational activity before they can be reconfigured.

Figure 2 shows this reconfigurable architecture according to the concrete syntax of the PervML[1] Domain-Specific Language for Smart Environments. Services are represented by circles, and Devices are represented by squares. Finally, the channels among services and devices are depicted by lines.

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

To specify which components and channels of the Smart Hotel support a certain feature, we use a weaving model [12]. Weaving models are used to define and to capture relationships between models elements by means of the WLink concept. A WLink express a link between model elements that has simple linking semantics. Its semantic has to be refined according to the use of the weaving model.

In our case, a WLink in the weaving model indicates that a given element in the PervML model will be included in the resulting PervML configuration if and only if a particular feature of the feature model is active. For instance, the blue color-coded mappings between features and PervML elements of Figure 2 are described by four WLinks as follows: Lighting by Occupancy wlink channel a, Lighting by Occupancy wlink channel b, Lighting by Occupancy wlink Presence Sensor 1 and Lighting by Occupancy wlink Presence Sensor 2. That is, by means of the weaving model, the PervML configuration is instantiated through the activation/inactivation of features in the Feature Model.

In order to query a weaving model to identify which PervML elements support a certain feature the Superimposition operator ($\odot$) is defined. The Superimposition takes a Feature and returns the set of components and channels related to this Feature. Some examples of the relationship between Features and the Smart Hotel (see Fig. 2) are as follows:

$$\odot(LightingByOccupancy) = \{a, b, 1, 2\},$$
$$\odot(OccupancySimulation) = \{c, 3, d\}$$

For example, Lighting by Occupancy is supported by the channels that are labeled as *a* and *b* and the components that are labeled as *1* and *2* as Fig. 2 shows.

Context Modelling We use an ontology-based context model that leverages Semantic Web technology and OWL (Web Ontology Language) [9]. OWL is an ontology markup language that enables context sharing and context reasoning. In the artificial intelligence literature, an ontology is a formal, explicit description of concepts in a particular domain of discourse. It provides a vocabulary for representing domain knowledge and for describing specific situations in a domain. An ontology-based approach for context modeling lets us describe contexts semantically and share common understanding of the structure of contexts among

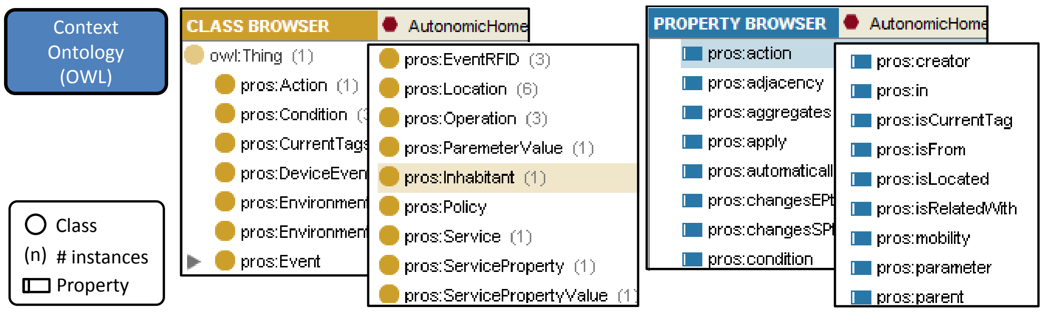


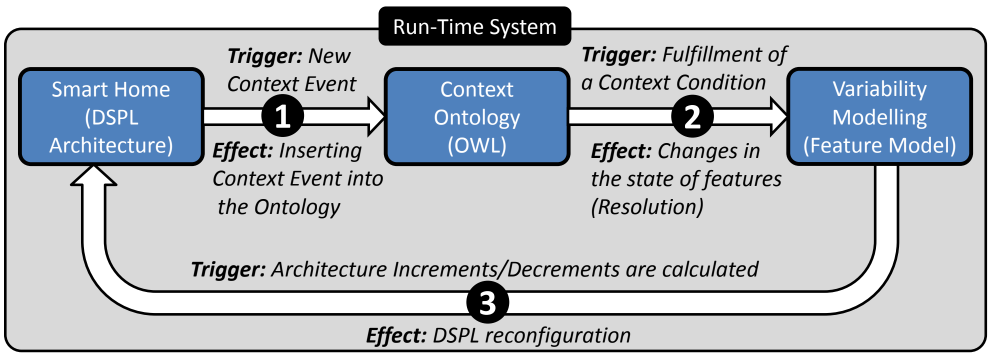
Figure 3: OWL Ontology for Smart Hotel.

users, devices, and services. The main benefit of this model is that it enables a formal analysis of the domain knowledge, such as performing context reasoning using first-order logic.

An ontology represents the DSPL context model structure. The ontology is described in OWL as a collection of RDF triples, in which each statement is in the form of (subject, predicate, object). The subject and object are the ontology objects or individuals and the predicate is a property relation defined by the ontology. For instance, (John, Location, Garden) means that John is located in the garden. Figure 3 shows our current ontology for context modelling in Smart Hotels. For more information about this ontology see [31].

The combination of the above techniques assists the DSPL prototype to determine the steps that are necessary to reconfigure itself. In particular, the prototype can activate/deactivate its own features dynamically at run-time according to the fulfillment of Context Conditions. The feature model specifies the possible configurations of the system, while the Dynamic Product Line Architecture can be rapidly retargeted to a specific configuration as follows (see Fig. 4).

The first step of the Reconfiguration Process is triggered whenever a context event is raised. For each conext event, the DSPL feed the Ontology with a RDF triple. Then, the DSPL checks if according to the new state of the Ontology any of the context conditions are fulfilled. The context conditions check for values in this ontology. For instance, the *Empty-Room* condition is fulfilled when none of the presence detection sensors is perceiving presence. This can be used to trigger the activation of



Run-Time System
Smart Home (DSPL Architecture)
Context Ontology (OWL)
Variability Modelling (Feature Model)
Trigger: New Context Event
1
Effect: Inserting Context Event into the Ontology
Trigger: Fulfillment of a Context Condition
2
Effect: Changes in the state of features (Resolution)
Trigger: Architecture Increments/Decrements are calculated
3
Effect: DSPL reconfiguration

Figure 4: Overview of the model-based reconfiguration process.

both the *In Room Detection* and the *Occupancy Simulation* features when all the inhabitants leave room. We can also define another context condition, *Comfort*, to trigger the activation of features related to ease and well-being such as *LightingbyOccupancy* or *PipedMusic*.

The second step of the Reconfiguration Process is triggered when a context condition is fulfilled. Since a given condition can trigger the activation/deactivation of several features, we define the Resolution concept (R) to represent the set of changes triggered by a condition. A *resolution* is a list of pairs where each pair is conformed by a Feature (F) and the state of the feature (S). Each *resolution* is associated to a context condition and represents the change (in terms of feature activation/deactivation) produced in the system when the condition is fulfilled.

$$R \overset{\text{def}}{=} \{(F, S)\} \mid F \in [\![FM]\!] \land S \in \{Active, Inactive\}$$

For instance, the conditions *EmptyRoom* and *Comfort* are associated to the following resolutions:

$$\begin{aligned} R_{EmptyRoom} &= \{(OccupancySimulation, Active), (InRoomDetection, Active), \\ &\quad (LightingByOccupancy, Inactive)\} \\ R_{Comfort} &= \{(PipedMusic, Active), (LightingbyOccupancy, Active)\} \end{aligned}$$

The $R_{EmptyRoom}$ resolution means that, when the Smart Hotel senses that it is empty (condition), it must reconfigure itself to deactivate *Lighting by Occupancy* and to activate both *Occupancy Simulation* and *In Room Detection*.

The third step of the reconfiguration process (see Fig 4) addresses the architecture reconfiguration of the DSPL. In the $R_{EmptyRoom}$ example, the DSPL queries the Feature Model to determine the architecture for that specific context. The architecture increments and decrements are calculated in order to determine the actions that are necessary to modify the current configuration of the DSPL.

These increments and decrements indicate how system components should be reorganized for the reconfiguration in order to move from one configuration of the system (User in the room, see left side of Figure 2) to another configuration (Nobody in the room, see right side of Figure 2). As illustrated in Fig. 2, the presence sensors are no longer used for lighting (communication channels a and b are disabled), and they are used to provide information to the security service instead (communication channels e and f are enabled). In addition, the presence simulation service (labelled as 3) is activated, and the communication channels required for this service to communicate with multimedia (channel c) and lighting (channel d) are established.

3. Case Study: Smart Hotel DSPL

This section introduces the case study of a smart hotel, which reconfigures its services and devices according to changes in the surrounding context, as described in the previous section. The smart hotel was chosen as the reconfiguration-based case study for two main reasons: first, its nature as a shared environment in which different users use the same room over time. The clients each have their own preferences for the room, which should be adjusted to improve the quality of their stay; secondly, the preferences of the clients change depending on the activity performed (e.g., the clients usually have different preferences when they are watching a movie than when they are working).

Overall, the smart hotel case study describes the stay of one client in different scenarios. This includes the check-in process and the way the room interacts with the client and changes its features depending on the client activities in order to make the stay as pleasant as possible. To give an idea of the dimensions of the case study, we present the following metrics:

According to the Feature Modelling technique, the Smart Hotel presents **thirty nine Features**. Some examples of these features are the Temperature Control feature, which offers a heating and cooling system;

the Device Synchronization feature which synchronizes the devices that the user can have (e.g., laptop, mp3 player, or PDA) or the Security feature, which secures the room when the user is absent.

The main concepts of the Smart Hotel DSPL architecture are Services, Devices, and the Communication Channels among them. The Smart Hotel has **thirteen Services, twenty Devices and thirty-five Channels**. For instance, the Multimedia Service can establish communication channels to devices such as PDAs or MP3 players.

In the Smart Hotel, users can perform different activities. Specifically, our case study addresses **eight Scenarios**. These scenarios are: Check-in, Entering the Room, Working, Watching a Movie, Sleeping, Leaving the Room, Housekeeping and Check-out.

Detailed documentation about this case study is publicly available online at `http://www.carloscetina.com/papers/smart-hotel.pdf`.

3.1. Reconfiguration Scenarios of the Smart Hotel

This section provides a brief description of all the scenarios that make up the Smart Hotel DSPL. These scenarios cover possible situations that can occur in the smart room of a hotel. The descriptions also indicate the goal of each scenario from the point of view of reconfiguration.

Check-In. When the user registers (online from the internet or at the hotel's reception desk), he is provided with a wizard that makes a few questions to set up the room according to his preferences.

Goal: To reconfigure the room according to the preferences of each user.

Entering the room. When the user enters the room, the smart room detects all the devices that the user is traveling with.

Goal: To integrate the functionality of the user's devices with the room services.

Activity. The room reconfigures itself according to the activities that the user performs in it. The activities can be working, watching a movie or sleeping.

Goal: To reconfigure the room services according to the specific activity that the user is performing at any given moment.

Leaving the room. When the user leaves, the room is reconfigured to disable the services that are no longer needed. Because no one is in the room, it is reconfigured to save energy. The room takes into account when the user has planned to come back (agenda) so that the room is the conditions preferred by the user (illumination and temperature).

Goal: To save energy while there are no users in the room without disturbing them when they come back.

Housekeeping. The room is reconfigured to guarantee the user's privacy when the cleaning service is working in the room. All displays where personal information of the client can be obtained (e.g., TV) are disabled to guarantee privacy.

Goal: To guarantee the user's privacy when the user is not in the room but the hotel staff is.

Check-Out. Finally, when the user finishes the stay in the room, the smart room stops being personalized for that user and its services are reconfigured in order to save energy.

Goal: To reconfigure the room to energy-saver mode for the periods when there is no user using it.

By combining the scenarios introduced above in different ways, we can describe a user's stay at the Smart Hotel. For example, the user checks in the hotel (Check-in scenario) at the reception desk. When he receives his room card, he can immediately enter his room. When he enters the room (Entering the room scenario), he has some free time and he decides to watch a movie selecting one from the hotel's pay-per-view service (Watching a movie scenario). Since it is late, after watching the movie, the user decides to go to sleep (Sleeping scenario). The next morning, the system wakes him up at the time that he has scheduled. The user leaves the room (Leaving the room scenario). During the user's absence, the hotel's cleaning service performs the room's maintenance (Housekeeping scenario). When the user comes back (Entering the room scenario), he has to pack everything to return home. When everything is prepared, he leaves the room (Leaving the room scenario) and then checks out at the hotel's reception desk (Check-out scenario).

The above description is just one of the possible users stays at the Smart Hotel. Combining the scenarios of the case study is possible to describe other users stays as follows:

- Stay 1: Check-in → **Entering the Room** → **Watching a Movie** → Sleeping → Leaving the room
- Stay 2: Check-in → Entering the Room → Leaving the room → **Entering the Room** → **Watching a Movie**

Both Stay 1 and Stay 2 involve a reconfiguration from Entering the Room to Watching a Movie. However, there is one previous reconfiguration (depicted as →) in the case of Stay 1, and there are three previous reconfigurations in the case of Stay 2. Since the previous configurations may impact the result of the current reconfiguration (for instance the context model stores the events history, see Section2), we are interested in performing reconfigurations in the context of stays as real as possible (by means of humans participants) instead of preparing pairs of a trigger and expected reconfiguration.

4. Experimentation Logistics of the Smart Hotel DSPL

In this Smart Hotel DSPL, we are concerned with reliability-based risk of the run-time reconfigurations. This reliability-based risk depends on the probability that the DSPL reconfiguration will fail in the operational environment and the adversity of that failure. For the purpose of this work, we have adopted the definition in [37], which defines risk as a combination of two factors: probability of malfunctioning (Availability) and the consequences of malfunctioning (Severity). The probability of failure depends on the probability of the existence of a fault combined with the possibility of exercising that fault. Whereas a fault is a feature of a system that precludes it from operating according to its specification, a failure occurs if the actual output of the system for some input differs from the expected output [37].

It is difficult to find exact estimates for the probability of failure of individual components in the system. In this paper, we adopt the severity classification used in [26] (see Table 1). Therefore, we use a coarse-grained scale, defined as high (H), middle (M), and low (L). We did not adopt an ordinal scale (e.g., 1 to 5) because the values do not truly represent the differences between scales in ratio or distance. In fact, the differences in their values only give indications of their relative rankings. If needed, the scaling definition can be refined later to be more fine-grained or an ordinal scale can be used.

Attribute	**High**	**Middle**	**Low**
Availability	No single point failure	Only one single point of failure	The number of single points of failures > 1
Failure Severity	(aka **critical**) A failure may cause major system damage or loss of production.	(aka **margin**) A failure may cause minor system damage, delay, or minor loss of production.	(aka **minor**) A failure may not cause system damage but will result in unscheduled maintenance or repair.

Table 1: Scale Definition of Reliability Metrics.

4.1. Participants and Training

The participants were 5th-year computer engineering students at the Technical University of Valencia, Spain. In order to motivate the participants, the experimental tasks were course assessment tasks. However, the participants were explicitly not advised that the assessment tasks were part of a formal experiment in order to avoid any spurious effect as a result of the participants being aware of being studied (i.e., avoiding the "Good Subject" effect [30]).

For training purposes, there were two lectures (2 hours each) covering the main concepts of a DSPL and introducing the Smart Hotel DSPL. They also received training on the use of MoRE [4], the Model-based Reconfiguration Engine of the DSPL that supports the case study. MoRE was also used on other course related assignments.

4.2. Challenges to Introduce Human participants in DSPL Evaluation

Two major challenges were identified and addressed with the involvement of human subjects in the DSPL evaluation. DSPL reconfigurations are triggered by context events, many of which are difficult to reproduce in practice (e.g., a fire). To successfully evaluate DSPLs, we must **enable participants to trigger those reconfigurations that are relevant for the experimentation**, not only those reconfigurations that can be easily triggered.

When reconfigurations are performed some of the effects can be easily perceived (e.g., an alarm is triggered) while others are not (e.g., some sensors

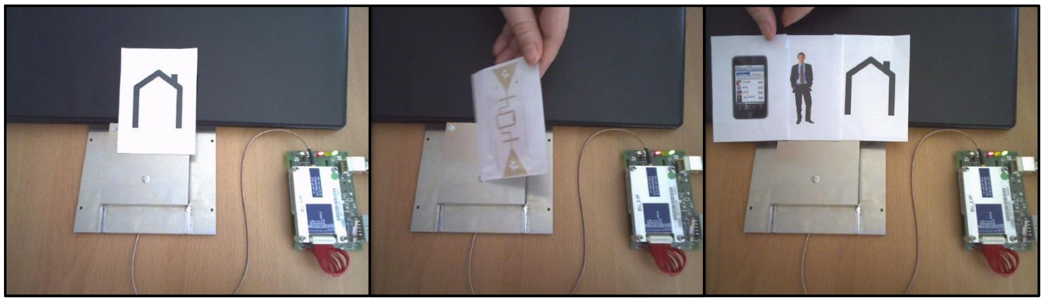


Figure 5: Context Cards for triggering DSPL reconfigurations.

are deactivated). To successfully evaluate DSPLs, we must **enable participants to understand and evaluate the effects of reconfigurations**. If participants misunderstand reconfiguration effects, they will not be able to apply the classifications scales of Availability and Severity.

4.2.1. Enabling Participants to Trigger Reconfigurations

Reconfigurations in the case study are triggered by different environmental conditions. When participants are experimenting with the reconfiguration scenarios, they should be able to reproduce these situations in order to validate the system reaction. Since many context events are difficult to reproduce in practice (e.g., simultaneous events that occur in different rooms or there is fire in the room), simulating them is a must.

The control of context events is essential for the evaluation of DSPLs, since context changes are the events that drive the reconfiguration of the DSPL. Mechanisms should be provided to users to allow them to easily change the current context of the system. In this way, users can move from one configuration to another configuration by applying context changes.

In order to provide an intuitive representation of context events that users could manipulate easily, we provided them with cards that depicted these events. The use of the card metaphor was chosen since it is a familiar concept for most people [35].

Each *context card* represents a context event (such as “phone ringing”). During evaluation sessions, the users were given a deck of context cards. The deck included the events that could affect the particular DSPL being evaluated. The users could then make use of the context cards as the building blocks for triggering the reconfiguration of the DSPL.

The design of the context cards was driven by the elements defined in the Smart Hotel ontology. Each card involved a specific instantiation of a class from the ontology. The information provided in the card included the type element and, optionally, some relevant attributes regarding its particular instantiation (such as the location where the event takes place). When the cards were designed, we tried to avoid including too much information. Thus, the users could easily recognize the different cards at a glance (see Fig. 5, right).

In order to automate the evaluation process, the Context Cards were enhanced with RFID tags (see Fig. 5, left). When a card is placed on the table it is automatically detected by an RFID antenna, and the context ontology is updated accordingly. In this way, the cards can be easily manipulated as if it was part of a card game. Furthermore, they are also closely integrated with the DSPL reconfiguraton engine (MoRE). That is, setting a context card close to the RFID antenna triggered the different reconfigurations by means of MoRE.

During the evaluation, the users could add and remove multiple cards from the table in order to define a specific context. The reconfiguration engine reconfigured the DSPL to fit the new context as it changed. Thus, the users could observe how the DSPL was reconfigured as they changed the context events.

The use of context cards enables users to evaluate the reaction of the system in different combinations of context events. Furthermore, putting users in control of the context definition provides valuable feedback. During our evaluation sessions, the users suggested new context cards and specific reconfigurations for certain context combinations that had not been previously considered by designers. Some new context cards were designed to group different events on a single card. Thus, a single card could represent the instantiation of several elements of the Smart Hotel ontology. This simplifies the activation of multiple conditions for users.

4.2.2. Enabling Participants to Evaluate the Reconfigurations

According to Dey in [10], one of the biggest challenges to the usability of context-aware applications (as is the case of a DSPL such as [24, 20, 36, 34, 25, 29]) is the difficulty that users have understanding why the applications do what they do. Dey defines the *intelligibility* concept as the support for users in understanding, or developing correct mental models of what a system is doing. This is done by providing explanations of why the system is taking a particular action and supporting users in predicting how the system might

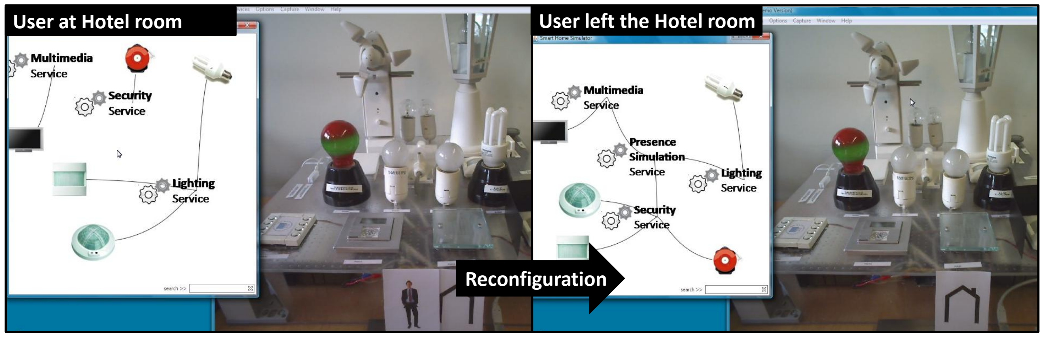


Figure 6: Visualizing reconfiguration effects by means of the Configuration Viewer.

respond to a particular input.

Since the DSPLs that we are developing are context-dependant, intelligibility becomes a challenge for their evaluation. When the Smart Hotel is reconfigured, some of the consequences are easily perceivable by users (e.g., an alarm is triggered) while others are not (e.g., some sensors are deactivated). Thus, we considered that the direct observation of the physical devices by the user is not enough for evaluating the DSPL reconfigurations. Mechanisms are required by users to allow them to fully understand the reconfiguration consequences (e.g. changes that are produced in rooms where the user is not present, etc.).

For the evaluation process a Configuration Viewer has been developed to provide users with visual information about the reconfiguration effects in the system. This tool provides a graphical representation of the relevant entities in the Smart Hotel room. These entities include the devices, services, and communication channels among them. When a context condition is activated, it is also depicted in the Configuration Viewer. Thus, the user can easily perceive that motion sensors are enabled and provide information to the alarm system when the room becomes empty. Without the Configuration Viewer, users cannot be sure whether or not the presence detection has been turned on when they leave the room. As Fig. 6 shows, direct observation of the physical devices is not enough to evaluate run-time reconfigurations.

Since we are interested in the evaluation of DSPL reconfigurations, it is not enough to represent the Smart Hotel room in a single state. Therefore, complementary information is provided to the users through our tool to depict what has changed from the previous configurations. By clicking on services or devices, the users get detailed information indicating changes in

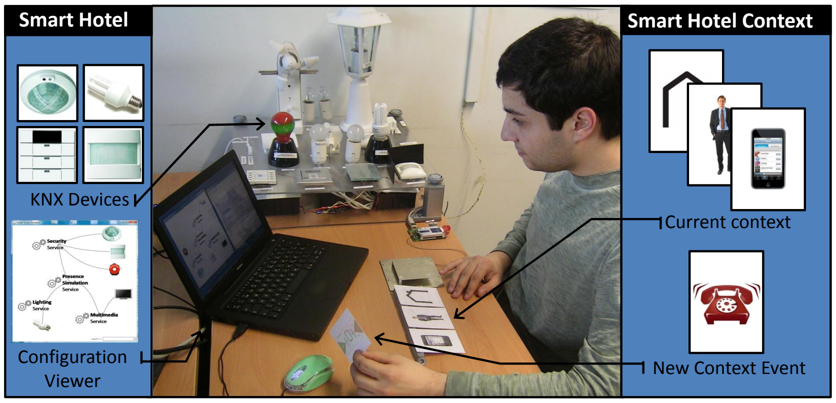


Figure 7: Experimentation set-up.

the configuration (e.g., the motion sensors provide the user with the following message: “motion sensors no longer in use to control lighting, currently in use to control security.”).

This use of the visualization tool enabled users to provide more accurate feedback during the DSPL evaluation since they could determine what has actually changed.

4.3. Experiment Operation

In the experimental set-up, a scale environment with real devices was used to represent the Smart Hotel. Therefore, the participants could interact with the same devices that can be found in a real deployment (see Fig. 7, top-left). The Configuration Viewer was used during the experiments to keep track of the system evolution. This tool graphically depicts the devices, the services, and the connections among them that are present in the system at any given moment (see Fig. 7, bottom-left). Since the reconfigurations are performed as a response to context events, mechanisms are provided for triggering them. We adopted RFID cards to set the Smart Hotel context (see Fig. 7, right). Each of the cards symbolized context information such as the presence of users or the occurrence of different events. These cards were combined to insert events in the ontology and to trigger reconfigurations in the Smart Hotel.

During the experiment, the same user interaction with the environment (activating a presence detector) produced different results according to the current configuration of the system (which depended on the context expressed by the cards). For example, an initial scenario could consist of a room where one inhabitant is present. The cards that defined this scenario are the ones illustrated in Fig. 7. In this scenario, the system architecture was organized in such a way that the piped music was available and the presence sensors were used by the lighting service. The user of the prototype could listen to the music and the lights were turned on/off as the user interacted with the sensors. If the card that represented the hotel inhabitant was removed, the sensors were automatically no longer used for the purpose of light control but for security instead. As a consequence, when the user of the prototype interacted with the sensors again, the alarm went off (see this reconfiguration example online[2]).

The above description is a small example of the evaluation performance of the reconfigurations introduced by DSPLs. Detailed specifications of the configurations and reconfigurations that make up the case study can be found in `http://www.carloscetina.com/papers/smart-hotel.pdf`. There are several videos available about the reconfiguration of our prototype Smart Hotel at http://www.autonomic-homes.com.

4.4. Data Collection

After each reconfiguration of the case study, the participants answered a questionnaire. The questionnaire asked the participant to set the *Availability* and *Failure Severity* for each reconfiguration according to the Scale Definition (see Table 1). The participants also indicated the number of context cards that they used to trigger the reconfigurations and whether or not they used the configuration viewer. Finally, the participants answered two questions related to the resulting configurations: "Do you think that the provided reconfiguration is adequate for the context conditions?" and "Do you think that further customization is required to fit your particular needs?". These two questions required the participants to provide a short explanation.

Reconfiguration failures could be caused by not only run-time errors but also specification errors, design errors, implementation errors or operational errors. However, lines between the former causes are blurred, and users may not clearly distinguish between them. Hence, when something goes wrong it may be hard for the participants to correctly attribute it to any particular cause. The use of a Model Based Reconfiguration Engine such as MoRE en-

[2]`http://www.youtube.com/watch?v=OVtE_RFeEKo&fmt=22`

abled us to keep traces of each reconfiguration: context ontology, variability and architecture models (both before and after the reconfiguration) and also the effects of the reconfigurations in terms of increments and decrements as described in Section 2. These reconfiguration traces turn out to be valuable information to achieve a more careful examination of reported failures in order to discard those failures out of the scope of run-time reconfigurations.

5. Experimentation Results

According to the results of the case study, most reconfigurations (87%) were reported as high *Availability* (see Fig. 8, Tables 2 and 3 for detailed information). This is mainly due to the fact that the Smart Hotel DSPL combines the *Decentralized Control System Reconfiguration* Pattern [15] and the FaMa framework [1] (variability analysis). That is, the Smart Hotel DSPL validates the resulting configuration of each reconfiguration before it is actually performed. If the reconfiguration led to an invalid configuration according to the feature model, then the reconfiguration would not be performed. However, experimentation revealed that even though the configurations were validated in terms of features, some of them went wrong in terms of the devices, services, or channels that make up the resulting configuration.

Single points of failure (9% + 4%) were identified mainly on devices and services that were not properly set up in the resulting configuration. In other words, some of these components remained in the old configuration when they were not supposed to, and others changed to a new configuration when they were not supposed to. Several subjects specifically reported that the configuration viewer eased the task of identifying these points of failure. In fact, 92% of the participants made use of the configuration viewer. However, they also reported that, in most of the cases, they double-checked the viewer by means of direct interaction with the smart devices and services. It was not until almost all of the scenarios were completed that most subjects began to fully trust the configuration viewer.

Overall, the DSPL reached a high level of *Availability* in most of the case study reconfigurations. However, experimentation revealed that even though DSPLs make use of run-time validation, they are not completely free of reconfiguration failures. To address this issue, we suggest complementing DSPLs with configuration viewers to help users easily detect points of failure.

With regard to the *failure severity*, few failures (8%) were indicated as critical (high severity). These critical failures were mostly related to services

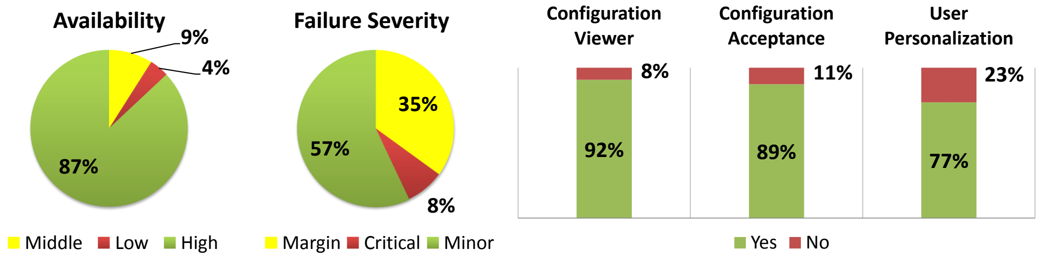


Figure 8: Overall results from the Case Study.

that provide inputs to other services. For instance, the *Presence Service* provided inputs to the following services: *Temperature, Multimedia, and Illumination*. In practice, the lines between producer and consumer services were blurred, and the subjects could not clearly distinguish between them. Hence, when something went wrong it was hard to correctly attribute it to just one specific service. Therefore, the subjects perceived that several services were malfunctioning at the same time. This suggests that services of this kind require more development resources (e.g. testing, quality control, etc.), since they affect the overall perception of the system.

With regard to *context cards* for triggering reconfigurations, Table 3 indicates the number of cards that were normally used to trigger the case study reconfigurations (minimum and maximum are shown as an indicator). The subjects did not reported any problems related to the understanding of these context cards. In fact, the subjects not only reported new combinations of the current context cards that should have their own reconfigurations, they also suggested new context cards and reconfigurations for these cards. The context card technique has provided us with interesting insights into the understanding and expectations that users have about reconfigurations. Context cards have not only proven to be a successful technique for setting the context for reconfigurations, but also for capturing reconfiguration requirements. Therefore, we suggest using this technique for both evaluation and requirements elicitation in DSPLs.

With regard to the *configuration acceptance*, we asked the users whether or not they considered the system reaction to be adequate taking into account the defined context events. Acceptance for the reconfiguration scenarios was high (89%). Most of the users considered behaviour provided to be a good response to the context defined with the cards, but they also considered that there was still room for improvement (as illustrated by the user personalization factor).

With regard to the *user personalization factor*, the users were asked whether or not they would modify the system reaction to better fit their needs. Since the specific needs of each user were very diverse (sometimes responding to opposite criteria), we identified the scenarios that could require more fine-grained reconfiguration capabilities. The subjects suggested configuration changes that were important and personally beneficial to them. They transformed configurations from conventional to personal. However, we do not believe that it is economically realistic to build specific features that individually suit participants. 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. In fact, the collected data supports that, although participants would modify the case study configurations (77%), most of them thought that the provided reconfiguration is adequate for the context conditions (89%).

6. Lessons Learned

Based on our experiences from this case study, we present the lessons that we learned to assist researchers in the context of DSPLs.

6.1. Introducing User Confirmations to Reconfigurations

During the evaluation of our DSPL, some subjects reported that they had triggered unintended reconfigurations by mistake. In other words, they mistakenly set up the context for one reconfiguration scenario (i.e. *EnteringTheRoom - LeavingTheRoom*), when they really wanted a different reconfiguration scenario (i.e., *EnteringTheRoom - Working*). Unintended reconfigurations of this kind were not counted as DSPL failures since they were human mistakes. However, this behaviour raised an interesting point regarding whether or not a reconfiguration should be confirmed before its execution.

After analyzing the unintended reconfigurations performed in our case study, we realized that they can be classified into three different categories. These categories take into account the implications of returning to the source configuration. The three categories are the following:

Round-trip. If there is a reconfiguration that leads directly to the source configuration from the unintended configuration, then we classify the

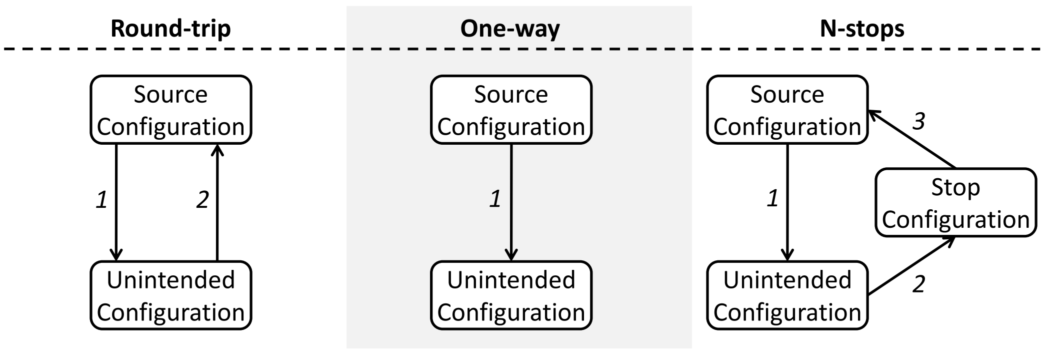


Figure 9: Categories for confirmation of reconfigurations.

reconfiguration as a round-trip one (see Figure 9, left). In our case study, some subjects performed unintended round-trip reconfigurations between *EnteringTheRoom* and *LeavingTheRoom* configurations. For these unintended round-trip reconfigurations, the subjects did not require any special support since they could easily find the way to return to the source configuration. In fact, most of the reconfigurations were not reported as unintended ones in our case study, and those that were reported as unintended did not require support to find the way back. Based on this experience, we do not think that DSPLs should ask for user confirmation before performing a round-trip reconfiguration.

One-way. If there is no reconfiguration that leads directly (or indirectly) to the source configuration from the unintended configuration, then we classify the reconfiguration as a one-way one (see Fig. 9, center). In our case study, some of the subjects performed unintended one-way reconfigurations between the *LeavingTheRoom* and *Check-Out* configurations. For these unintended one-way reconfigurations, the subjects always required support since they could not find a way back to the source configuration. Based on this experience, we suggest that DSPLs should ask for user confirmation before performing a one-way reconfiguration. This suggestion comes from the fact that once a one-way reconfiguration has been performed, it is not possible to find a way back to the source configuration.

N-stops. If there is a set of of reconfigurations that leads to the source configuration from the unintended configuration, then we classify the reconfiguration as a N-stops one (see Fig. 9, right). In our case study, some of the subjects performed unintended N-stop reconfigurations between the *EnteringTheRoom* and *Activity* configurations. For these unintended N-stops reconfigurations, almost all the subjects could easily find the way to return to the source configuration. However, a few subjects took a long time to find the way back. Based on this experience, we suggest that DSPLs should ask for user confirmation before performing an N-stops reconfiguration when the number of stops exceeds a certain limit. The purpose of our suggestion is to only require confirmation for critical reconfigurations. We also suggest identifying the acceptable limit of stops by applying Considerate Computing [14] techniques. These techniques take into account the domain particularities of the DSPL in order to determine when the number of reconfiguration stops is not trivial.

Since unintended reconfigurations can occur in DSPLs driven by context events [24, 20, 36, 34, 25, 29] or by user actions [16], we believe that confirmation patterns defined in this study can help DSPLs engineers to mitigate the unintended reconfigurations. Furthermore, we think that these confirmation patterns are specially relevant for DSPLs driven by context events, since users of these DSPLs usually do not control all the feasible context events and can miss a specific configuration because of it. The confirmation guidelines that came from our case study experience can contribute to avoid this kind of undesired behaviour.

6.2. Improving Reconfiguration Feedback

When the subjects of the study perceived the effects of a specific reconfiguration, they sometimes noticed that the result was not the expected one. In those cases, they indicated the presence of a reconfiguration failure, and they also evaluated the severity of the failure. One of the main issues with the evaluation process was related to the termination of the reconfigurations.

Since, each reconfiguration involves changes in different devices, services or communication channels, a delay between the event and the system reaction is introduced. This delay varies from reconfiguration to reconfiguration. Some subjects reported that it was difficult for them to determine whether the reconfiguration process was completed or there were still actions pending. This could lead to misidentifying failure or to misevaluating severity, since a subject could start evaluating a reconfiguration before it was actually finished.

To address this issue, our configuration viewer was enhanced with notification messages that indicated the completion of each reconfiguration. The subjects were provided with feedback regarding the overall process as well as at the service/device level. When a service or device was in the process of reconfiguration, it was depicted as busy (a waiting icon) in the configuration viewer.

Most of the subjects reported that they found this reconfiguration feedback to be very useful not only for failed reconfigurations but also for regular reconfigurations. Therefore, we suggest that DSPLs should provide feedback about the termination of reconfigurations, especially, when reconfigurations involve human users.

6.3. Introducing Rollback Capabilities to Reconfigurations

Our case study raised another important concern in connection with DSPL recovery after a failure. Once a reconfiguration failure was identified and evaluated, a few subjects required support to resume the experimentation. They reported problems in performing the next reconfiguration after the failure. In other words, they did not find a simple way to reach another configuration of the case study. Below, we present the main kinds of issues reported and how we think they should be addressed in DSPLs.

Unexpected configurations. After a failure reconfiguration, a few subjects reported that the resulting configuration was not the expected one. In place of the expected configuration (i.e., *WatchingAMovie*), they got another configuration (i.e., *Working*). In most of these cases, the subjects could perform a new reconfiguration in order to reach the expected configuration. However, a few of the cases required several reconfigurations to reach the expected configuration. To address this issue, in DSPLs, we suggest introducing some sort of "undo" operation that returns the system directly to the previous configuration.

This has several implications for the design of DSPLs since some actions have collateral effects that cannot be easily undone (e.g., sending an e-mail). The handling of compensation actions to reverse a reconfiguration should be studied, also the consequences of a rollback need to be explained so that users can be provided information to help them choose among different compensation actions and understand how they relate to their desired goals.

Unknown configurations. After a failure reconfiguration, some subjects reported that they failed to identify the resulting configuration in the Smart Hotel documentation. In other words, the resulting configuration was different from all the documented configurations that made up the case study. The Feature Model of the Smart Hotel defines more configurations than the ones considered in our case study. These *unknown configurations* imply that the subjects could not identify the set of reconfigurations that led to the expected configuration. Therefore, they needed support to continue the experimentation. To address this issue, we strongly suggest an "undo" operation that returns the system directly to the previous configuration. Note that for *Unknown configurations*, we think that the "undo" operation should be mandatory. However, for *Unexpected configurations*, we think that the "undo" operation should be optional since users have an alternative to achieve the expected configuration.

The DSPL that supports this case study makes use of Feature Models at run-time to determining how to perform the reconfigurations. According to a recent discussion on DSPL architectures [6], other DSPL approaches make use of different techniques to perform reconfigurations (i.e., QoS properties or UML profiles). Although the details are different, these DSPLs are based on variability specifications, and their reconfiguration can also lead to *Unexpected configurations* or *Unknown configurations*. Even though these DSPLs could achieve an expected configuration from any given *Unexpected* or *Unknown* configuration, our experience suggests that introducing an "undo reconfiguration" operation is simpler and more practical from the viewpoint of the DSPL user.

Overall, the main motivation of the rollback capabilities is to keep the user in control of the reconfigurations, no matter the reconfigurations are triggered by mistake or intentionally. That is, whenever the user disagrees with the resulting configuration he or she has the option to undo the reconfigurations. For instance, even though the user goes to sleep and the room reconfigures (correctly) to the sleeping configuration, if the resulting configuration does not match the user expectations, he or she can go back to the previous configuration.

7. Related Work

Since DSPL architectures are retargeted to different configurations at run-time, they could benefit from current approaches for adaptive architecture evaluation. Specifically, Yacoub and Ammar [37] proposed a method for reliability risk assessment at the architecture level. This method is based on component-based systems in which implementation entities explicitly invoke each other. Liu et al. [26] also proposed a method for evaluating reliability by means of fault tolerance and fault prevention. They identified architectural design patterns to build an adaptive architecture that is capable of preventing or recovering from failures. In comparison with our work, these methods do not address runtime reconfigurations that are driven by variability specifications such as Feature Models. However, we have succesfully applied some of these proven techniques (such as estimation of availability and severity) to the evaluation of DSPLs.

For SPL evaluation, several approaches have produced results in connection to quality properties such as reliability. For example: the extended goal-based model [18], the F-SIG Feature-softgoal interdependency graph [22], the Benavides et al. [1] approach, Zhang et al. [38] Bayesian Belief Network (BBN). There are also other methods that are not based on Feature Models: COVAMOF (ConIPF Variability Modelling Framework) [32] and Quality Requirements of a Software Family (QRF) method [27]. Most of these approaches usually remain at the Domain Engineering phase of SPLs only, they do not address run-time reconfigurations as our work does. Therefore, these approaches are not suitable for DSPL evaluation.

Other approaches address reliability evaluation of SPL products at runtime. The RAP approach [21] defines how the reliability requirements should be mapped to the architecture and how the architecture should be analyzed in order to validate whether or not the requirements are met. Etxeberria et al. [11] also take into account reliability at run-time and present a generic approach that can be combined with existing architecture evaluation methods such as PASA [23] or SALUTA [13]. However, both the RAP and Etxeberria approaches are oriented to *static products* only. Conversely, our work addresses the evaluation of *reconfigurable products* such as these in DSPLs. Furthermore, we address the challenges of evaluating reconfigurable products and we provide guidelines to improve the development of future DSPLs.

8. Concluding Remarks

With more and more devices being added to our surroundings, simplicity becomes greatly appreciated by users. Dynamic Software Product Lines (DSPL) encompasses systems that are capable of modifying their own behavior with respect to changes in their operating environment by using run-time reconfigurations. However, failures in these reconfigurations directly impact the user experience since the reconfigurations are performed when the system is already under user control. This is in contrast to SPLs where all the configurations are performed before delivering the system to the users.

Given the importance of run-time reconfigurations in DSPLS, we have evaluated the reliability-based risk of these reconfigurations, specifically, the probability of malfunctioning (Availability) and the consequences of malfunctioning (Severity). The evaluation has been performed by means of the Smart Hotel case study which was deployed with real devices with the participation of human subjects.

Furthermore, we successfully identified and addressed two challenges associated with the involvement of human subjects in DSPL evaluation. On the one hand, DSPL reconfigurations are triggered by context events many of which are difficult to reproduce in practice. To evaluate DSPLs, we successfully applied a technique based on Context Cards to enable participants to trigger reconfigurations. On the other hand, when reconfigurations are performed, some of the effects are easily perceived (e.g., an alarm is triggered) while others are not (e.g., some sensors are deactivated). For this problem, we successfully applied a technique to enable participants to understand and evaluate the effects of reconfigurations. If participants misunderstand the reconfiguration effects, they will not be able to apply the classification scales of Availability and Severity. We believe that these techniques can also contribute to the evaluation of more quality properties in the context of DSPLs.

The evaluation of the case study reveals positive results regarding both Availability and Severity. We hope that these positive results encourage researchers and practitioners to apply DSPL to other promising areas of research such as mobile devices or automotive systems. However, the participant feedback in this study highlihts issues with recovery from a failed reconfiguration or a reconfiguration triggered by mistake. To address these issues, we have provided some guidelines learned in the case study.

From the evaluation of the Smart Hotel DSPL, we consider the following as our key contributions:

- **The identification and solution of two challenges** associated with the involvement of human subjects **in DSPL evaluation**: to (1) trigger run-time reconfigurations and to (2) understand the effects of the reconfigurations. These techniques can be applied not only to reliability-based risk but also to other quality properties that require the execution of reconfigurations by human users, for instance, usability or security.
- **The experimentation results**, which reveal the maturity of run-time reconfigurations with regard to both Availability and Severity. These results can encourage researchers and practitioners to apply DSPL to other promising domains.
- **The identification and solution of key issues for user acceptance of DSPLs**: to (1) recover from a failed reconfiguration, and to (2) recover from a reconfiguration triggered by mistake.

Finally, we conclude that the DSPL has achieved satisfactory results regarding reliability-based risk; nevertheless, DSPL engineers must provide users with more control over the reconfigurations or the users will not be comfortable with DSPLs even though they achieve a high level of reliability.

Acknowledgments.

We are grateful to the participants of this case study. We also thank both Esteban Saiz and David Unio who helped in preparing the Smart Hotel Case Study. This work has been developed with the support of MICINN under the project EVERYWARE TIN2010-18011.

Appendix A

		Availability			**Failure Severity**		
#	**Reconfiguration**	High	Middle	Low	Minor	Margin	Critical
1	Check-In - Entering	95%	5%	0%	60%	39%	1%
2	Entering - Sleeping	86%	12%	2%	63%	33%	4%
3	Entering - Working	88%	9%	3%	65%	30%	5%
4	Entering - Movie	85%	13%	2%	56%	37%	7%
5	Sleeping - Working	85%	11%	4%	62%	30%	8%
6	Working - Sleeping	84%	8%	8%	65%	29%	6%
7	Sleeping - Movie	90%	3%	7%	61%	32%	7%
8	Movie - Sleeping	88%	11%	1%	64%	31%	5%
9	Working - Movie	87%	12%	1%	65%	29%	6%
10	Movie - Working	89%	10%	1%	68%	27%	5%
11	Sleeping - Leaving	86%	9%	5%	58%	34%	8%
12	Movie - Leaving	84%	15%	1%	67%	26%	7%
13	Working - Leaving	86%	9%	5%	56%	33%	11%
14	Entering - Leaving	85%	5%	10%	65%	29%	6%
15	Leaving - Entering	89%	0%	11%	63%	32%	5%
16	Leaving - H. Keeping	81%	17%	2%	18%	57%	25%
17	H. Keeping - Leaving	79%	15%	6%	9%	68%	23%
18	Leaving - Check-Out	96%	3%	1%	58%	39%	3%

Table 2: Results of the Case Study grouped by reconfigurations, part 1.

		# Cards		Viewer		Acceptance		Perso.	
#	**Reconfiguration**	Min	Max	Yes	No	Yes	No	Yes	No
1	Check-In - Entering	1	2	72%	28%	97%	3%	79%	21%
2	Entering - Sleeping	3	4	73%	27%	86%	14%	77%	23%
3	Entering - Working	4	4	89%	11%	84%	16%	74%	26%
4	Entering - Movie	3	4	92%	8%	90%	10%	71%	29%
5	Sleeping - Working	4	5	87%	13%	84%	16%	75%	25%
6	Working - Sleeping	3	4	94%	6%	86%	14%	77%	23%
7	Sleeping - Movie	4	5	95%	5%	90%	10%	78%	22%
8	Movie - Sleeping	3	4	92%	8%	86%	14%	82%	18%
9	Working - Movie	3	3	94%	6%	91%	9%	81%	19%
10	Movie - Working	3	5	96%	4%	84%	16%	77%	23%
11	Sleeping - Leaving	3	6	97%	3%	88%	12%	73%	27%
12	Movie - Leaving	3	4	97%	3%	87%	13%	72%	28%
13	Working - Leaving	3	4	96%	4%	88%	12%	67%	33%
14	Entering - Leaving	2	3	96%	4%	88%	12%	70%	30%
15	Leaving - Entering	2	3	97%	3%	96%	4%	78%	22%
16	Leaving - H. Keeping	2	3	99%	1%	86%	14%	87%	13%
17	H. Keeping - Leaving	2	3	98%	2%	87%	13%	85%	15%
18	Leaving - Check-Out	2	3	97%	3%	96%	4%	78%	22%

Table 3: Results of the Case Study grouped by reconfigurations, part 2.

References

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

[2] M. N. Bennani and D. A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, Washington. USA, 2005.

[3] 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.

[4] C. Cetina, P. Giner, J. Fons, and V. Pelechano. Autonomic computing

through reuse of varibility models at run-time: The case of smart homes. *Computer*, pages 46–52, October 2009.

[5] C. Cetina, P. Giner, J. Fons, and V. Pelechano. Using feature models for developing self-configuring smart homes. Fifth International Conference on Autonomic and Autonomous Systems (ICAS 2009), April 2009.

[6] 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.

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

[8] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. In *Proceedings of the Third Software Product Line Conference 2004*, pages 266–282. Springer, LNCS 3154, 2004.

[9] M. Dean and G. Schreiber. OWL web ontology language reference. W3C recommendation, W3C, February 2004.

[10] A. K. Dey. Modeling and intelligibility in ambient environments. *Journal of Ambient Intelligence and Smart Environments (JAISE)*, 1(1):57–62, January 2009.

[11] L. Etxeberria and G. Sagardui. Variability driven quality evaluation in software product lines. In *SPLC '08: Proceedings of the 2008 12th International Software Product Line Conference*, pages 243–252, Washington, DC, USA, 2008. IEEE Computer Society.

[12] D. D. Fabro. *Metadata management using model weaving and model transformation.* PhD thesis, University of Nantes, September 2007.

[13] E. Folmer, J. van Gurp, and J. Bosch. Scenario-based assessment of software architecture usability. In *Proceedings of Workshop on Bridging the Gaps Between Software Engineering and Human-Computer Interaction*, pages 61–68, 2003.

[14] W. W. Gibbs. Considerate computing. *Scientific American*, 292(1):54–61, 2004.

[15] H. Gomaa and M. Hussein. Dynamic software reconfiguration in software product families. In *Software Product-Family Engineering, 5th International Workshop*, pages 435–444, 2003.

[16] H. Gomaa and M. Hussein. Dynamic software reconfiguration in software product families. *Software Product-Family Engineering*, pages 435 – 444, 2004.

[17] H. Gomaa and M. Hussein. Software reconfiguration patterns for dynamic evolution of software architectures. In *4th Working IEEE / IFIP Conference on Software Architecture*, pages 79–88, 2004.

[18] B. Gonzalez-Baixauli, M. Laguna, and Y. Crespo. Product line requirements based on goals, features and use cases. *International Workshop on Requirements Reuse in System Family Engineering*, pages 4–6, 2004.

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

[20] S. Hallsteinsen, E. Stav, A. Solberg, and J. Floch. Using product line techniques to build adaptive systems. *Software Product Line Conference, 2006 10th International*, pages 10 pp.–, 21-24 Aug. 2006.

[21] A. Immonen. A method for predicting reliability and availability at the architecture level. In *Software Product Lines*, pages 373–422. 2006.

[22] S. Jarzabek, B. Yang, and S. Yoeun. Addressing quality attributes in domain analysis for product lines. *Software, IEE Proceedings -*, 153(2):61–73, April 2006.

[23] W. L. and S. C. Pasa a method for the performance assessment of software architectures. In *Proceedings of the Third International Workshop on Software and Performance (WOSP2002*, pages 179–189. ACM Press, 2002.

[24] J. Lee and K. Kang. A feature-oriented approach to developing dynamically reconfigurable products in product line engineering. In *Software Product Line Conference, 2006 10th International*, 2006.

[25] T. Lemlouma and N. Layaida. Context-aware adaptation for mobile devices. *Mobile Data Management, 2004. Proceedings. 2004 IEEE International Conference on*, pages 106–111, 2004.

[26] Y. Liu, M. A. Babar, and I. Gorton. Middleware architecture evaluation for dependable self-managing systems. In *QoSA '08: Proceedings of the 4th International Conference on Quality of Software-Architectures*, pages 189–204, Berlin, Heidelberg, 2008. Springer-Verlag.

[27] E. Niemelä and A. Immonen. Capturing quality requirements of product family architecture. *Inf. Softw. Technol.*, 49(11-12):1107–1120, 2007.

[28] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Real-Time Syst.*, 23(1-2):127–141, 2002.

[29] C. Parra, X. Blanc, and L. Duchien. Context Awareness for Dynamic Service-Oriented Product Lines. *Software Product Line Conference, 2009. SPLC 2009. 13th International*, 24-28 Agust. 2009.

[30] R. Rosnow and R. Rosenthal. People studying people: Artifacts and ethics in behavioral research. *W.H. Freeman and Company*, 1997.

[31] E. Serral, P. Valderas, and V. Pelechano. A model driven development method for developing context-aware pervasive systems. In *UIC '08: Proceedings of the 5th international conference on Ubiquitous Intelligence and Computing*, Berlin, 2008.

[32] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch. Covamof: A framework for modeling variability in software product families. In *SPLC*, pages 197–213, 2004.

[33] G. Tesauro, N. Jong, R. Das, and M. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, pages 65–73, June 2006.

[34] P. Trinidad, , A. Ruiz-Cortés, and J. P. na. Mapping feature models onto component models to build dynamic software product lines. *International Workshop on Dynamic Software Product Line*, 2007.

[35] M. J. Weal, D. Cruickshank, D. T. Michaelides, K. Howland, and G. Fitzpatrick. Supporting domain experts in creating pervasive experiences. In *Pervasive Computing and Communications, 2007. PerCom*

'07. Fifth Annual IEEE International Conference on, pages 108–113, March 2007.

[36] J. White, D. C. Schmidt, E. Wuchner, and A. Nechypurenko. Automating product-line variant selection for mobile devices. *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 129–140, 10-14 Sept. 2007.

[37] S. M. Yacoub and H. H. Ammar. A methodology for architecture-level reliability risk analysis. *IEEE Trans. Softw. Eng.*, 28(6):529–547, 2002.

[38] H. Zhang, S. Jarzabek, and B. Yang. Quality prediction and assessment for product lines. page 1031. 2003.