Informe Técnico / Technical Report




Case Study: Improving the Smart Hotel DSPL

María Gómez, Joan Fons, Carlos Cetina, Vicente Pelechano





Ref. #:	ProS-TR-2011-04				
Title:	Case Study: Improving the Smart Hotel DSPL				
Author (s):	María Gómez, Joan Fons, Carlos Cetina, Vicente Pelechano				
Corresponding author (s):	María Gómez, magomez@dsic.upv.es Joan Fons, jjfons@dsic.upv.es Carlos Cetina, ccetina@dsic.upv.es Vicente Pelechano, pele@dsic.upv.es				
Document version number:	1.0	Final version:	No	Pages:	18
Release date:	February 2011				
Key words:	DSPL design, Design properties, Refinements				

Universidad Politécnica de Valencia · Camino de Vera s/n · Edificio 1F · 46022 Valencia Spain ·
T. +34 96 387 70 07 Ext. 83530 · M. + 34 619 543 623 · F. +34 96 387 73 59 · info@pros.upv.es · www.pros.upv.es

Index

Figure Index

Table Index

1. Global description of the Smart Hotel Case Study

This document illustrates the application of a design method for improving design of Dynamic Software Product Lines (DSPLs) through systematic variability refinements. The design method has been applied in a previous case study, the Smart Hotel DSPL. Detailed documentation about this case study is publicly available online at http://www.carloscetina.com/papers/smart-hotel.pdf.

The Smart Hotel DSPL reconfigures its services and devices according to changes in the surrounding context. In the Smart Hotel, clients can develop different activities. The Smart Hotel changes its features (activating or deactivating these features) depending on the client's activities in order to make the stay as pleasant as possible. As a starting point, we use a reduced version of the original Smart Hotel DSPL specification in order to facilitate the understanding of the design method. The Smart Hotel DSPL uses variability models at run-time to drive the reconfigurations of the system: how a system can activate or deactivate its own features dynamically at run-time by fulfilling certain context conditions. According to the Feature Modeling technique, the Smart Hotel Feature Model (FM) presents 18 features (Figure 1).

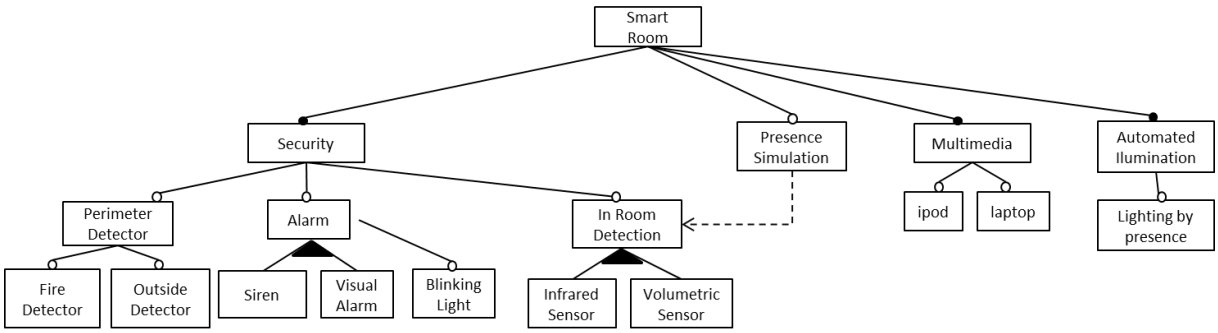

Figure 1. Feature Model describing the Smart Hotel DSPL case study.

A resolution translates contextual changes into changes in the activation/deactivation of features in the feature model. We have defined the following 4 resolutions:

RESOLUTION	CONDITION	DESCRIPTION	FEATURES	STATE
R1	cond1	A person is entering in the room	Automated Ilumination	True
			InfraredSensor	False
			LightingByPresence	True
			PresenceSimulation	False
R2	cond2	A person is leaving the room	Automated Ilumination	False
			InfraredSensor	True
			LightingByPresence	False
			PresenceSimulation	True
R3	cond3	Sleeping	InRoomDetection	False
			VolumetricSensor	False
R4	cond4	A person starts listening to music	iPod	True

Table 1. Initial set of resolutions.

The FM represents the different room configurations, while the resolutions represent the reconfigurations of the system at run-time. Let's suppose that the FM initially have the following features active:

- SmartRoom
- Security
- Alarm
- InRoomDetection
- VolumetricSensor

Given the feature model and the set of resolutions, the DSPL execution can be abstracted in a state machine model which represents the possibility space of the DSPL Smart Hotel. The possibility space represents all the feasible configurations that the system can reach because of the fulfillment of each resolution. Figure 2 illustrates the possibility space derived from the DSPL specification. The definition of 4 resolutions generates a possibility space with 12 possible configurations and 28 reconfigurations among them.

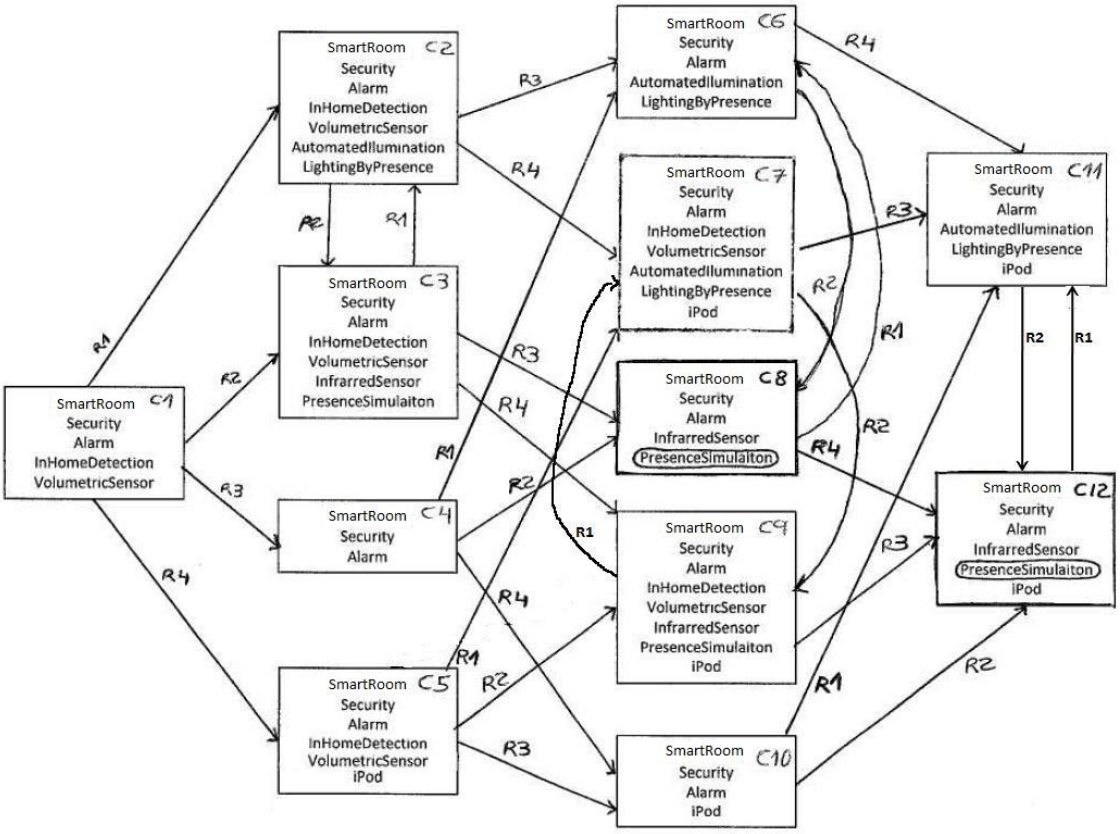

Figure 2. Possibility Space of the Smart Hotel DSPL.

Once the possibility space has been obtained, the designer can select different design properties in order to improve the DSPL design and consequently the DSPL execution.

2. Design Properties Catalog

This section presents the defined design properties. The properties have been grouped in categories. Figure 3 illustrates the design properties classified in form of a feature model.

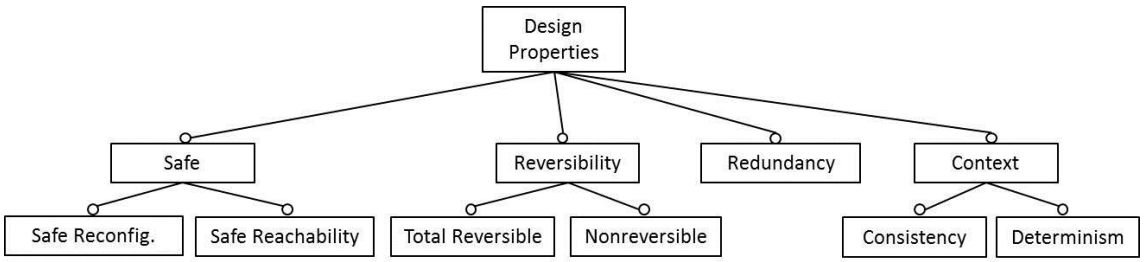

Figure 3. Design Properties Catalog

Next, we show the application of all the properties to improve the Smart Hotel DSPL.

a) Property 1: Safe Reconfigurations and Reachability

This property guarantees that after reconfigurations the system never reaches invalid states. Invalid states refer to inconsistent states where variability constraints are violated. The refinement associated to this property guarantees that all reconfigurations are safe (do not lead to invalid states) and that no configuration is reached through invalid states.

The FM (Figure 1) contains the following variability constraint:

PresenceSimulation ***requires*** InRoomDetection

By means of the analysis operations of FAMA framework [1], the configurations of the possibility space can be classified as valid or invalid, depending on whether or not the reconfigurations satisfy variability restrictions of the FM.

In the Smart Hotel possibility space (Figure 2) there are two invalid configurations C8 and C12, since in both configurations feature *PresenceSimulation* is active while feature *InRoomDetection* is inactive. Consequently, the input and output reconfigurations of C8 and C12 are unsafe reconfigurations and should be avoided from the specification. The possibility space contains 9 unsafe reconfigurations. Figure 4 illustrates the possibility space with invalid configurations and unsafe reconfigurations highlighted in color red.

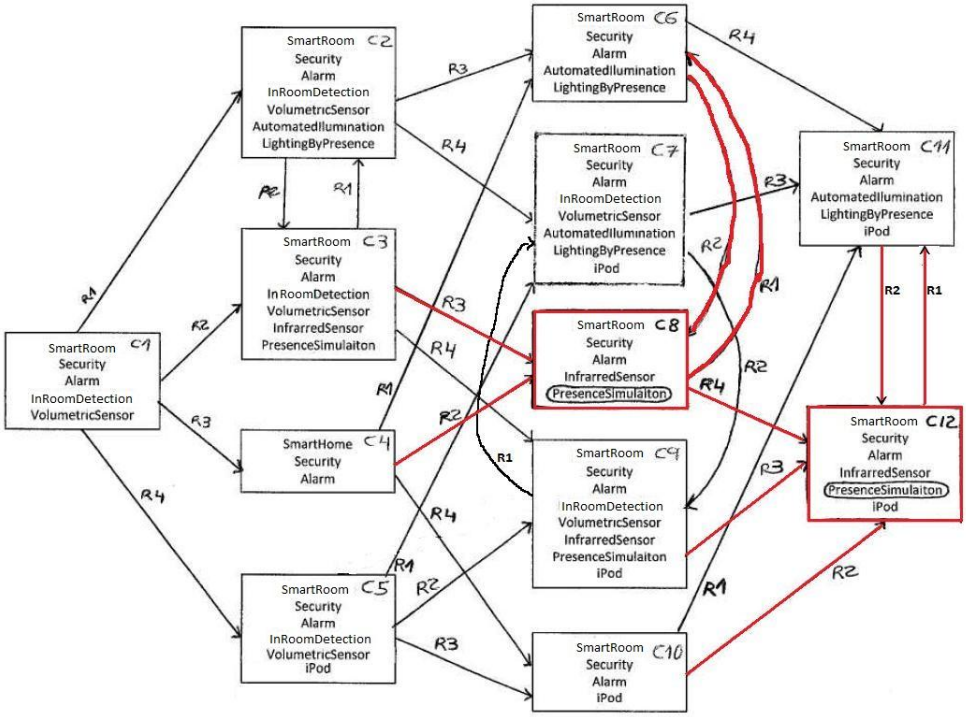

Figure 4. Possibility Space with invalid configurations and unsafe reconfigurations identified.

This property has two associated model refactorings to achieve a specification (1) free of unsafe reconfigurations and (2) free of unreachable valid configurations.

- **Refactoring 1: Safe reconfigurations**

The first refactoring modifies the resolutions in order to avoid unsafe reconfigurations. The refactoring introduces guards to those resolutions that trigger unsafe reconfigurations. The resolutions after applying the Safe Reconfigurations refactoring are shown in Table 2.

RESOLUTION	CONDITION	GUARD	FEATURES	STATE
R1	cond1	***[!C8 ^ !C12]***	Automated Ilumination	True
			InfraredSensor	False
			LightingByPresence	True
			PresenceSimulation	False
R2	cond2	***[!C4 ^ !C6 ^ !C10 ^ !C11]***	Automated Ilumination	False
			InfraredSensor	True
			LightingByPresence	False
			PresenceSimulation	True
R3	cond3	***[!C3 ^ !C9]***	InRoomDetection	False
			VolumetricSensor	False
R4	cond4	***[!C8]***	iPod	True

Table 2. Reconfigurations after applying Safe Reconfigurations refactoring.

- **Refactoring 2. Safe reachability**

The second refactoring modifies the resolutions in order to avoid unreachable valid configurations. Since the old unsafe reconfiguration lead to the current unreachable configuration, the refactoring calculates a new safe reconfiguration which directly leads to the unreachable configuration from the valid source of the old unsafe reconfiguration. It also introduces guards to the new resolution in order to avoid the generation of new configurations because of the application of this resolution from other configurations. This refactoring generates the new resolutions shown in Table 2.

RESOLUTION	CONDITION	GUARD	FEATURES	STATE
R3+R1	cond3^cond1	[C3 v C9]	InRoomDetection	False
			VolumetricSensor	False
			Automated Ilumination	True
			InfraredSensor	False
			LightingByPresence	True
			PresenceSimulation	False
R3+R4+R1	cond3^cond4^ cond1	[C3]	InRoomDetection	False
			VolumetricSensor	False
			iPod	True
			Automated Ilumination	True
			InfraredSensor	False
			LightingByPresence	True
			PresenceSimulation	False
R2+R1	cond2^cond1	[C4 v C10]	Automated Ilumination	False
			InfraredSensor	True
			LightingByPresence	False
			PresenceSimulation	True
			Automated Ilumination	True
			InfraredSensor	False
			LightingByPresence	True
			PresenceSimulation	False
R2+R4+R1	cond2^cond4^ cond1	[C4]	iPod	True
			Automated Ilumination	False
			InfraredSensor	True
			LightingByPresence	False
			PresenceSimulation	True
			Automated Ilumination	True
			InfraredSensor	False
			LightingByPresence	True
			PresenceSimulation	False

Table 3. Reconfigurations after applying Safe Reachability refactoring.

The Smart Hotel possibility space after applying the two refactorings above is shown in Figure 5.

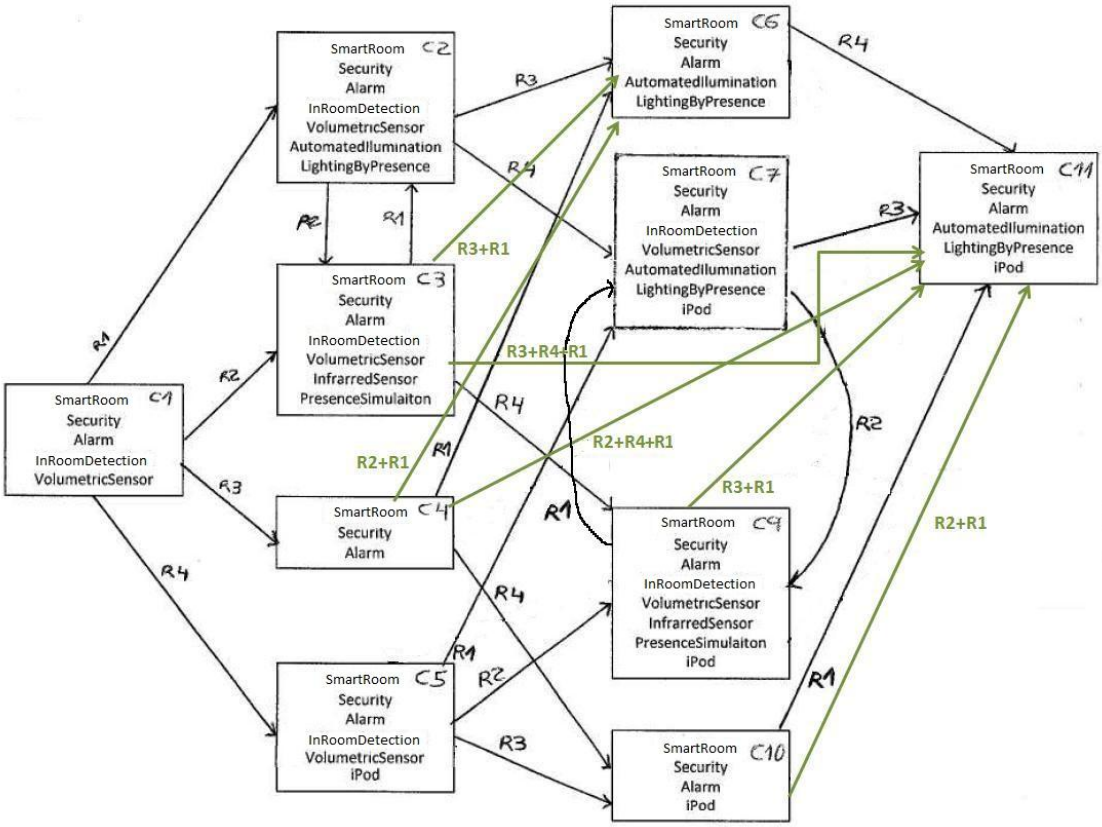

Figure 5. Possibility Space after applying Safe Reconfigurations and Reachability refactorings.

b) Property 2: Redundant Reconfigurations

This property guarantees that the specification does not contain duplicated behavior. We understand as redundancy to be the duplication of behavior. Two or more different resolutions are redundant if they produce the same effect on the system. That is to say, they generate reconfigurations that evolve the system from the same source configuration to the same target configuration by means of different actions.

It can be observed that in the last possibility space (Figure 5) the reconfigurations triggered by resolutions R1 and R2+R1 are redundant because both evolve the system between the same configurations C4-C6 and C10-C11. Table 4 shows the redundant reconfigurations.

RESOLUTION	SOURCE	DESTINATION	CONDITION	GUARD
R1	C4	C6	cond1	[!C8 ^ !C12]
R2+R1	C4	C6	cond2^cond1	[C4 v C10]
R1	C10	C11	cond1	[!C8 ^ !C12]
R2+R1	C10	C11	cond2^cond1	[C4 v C10]

Table 4. Redundat reconfigurations in the Possibility Space.

- **Refactoring 1: Redundancy Avoidance**

This refinement eliminates redundant reconfigurations in the DSPL. First, the redundant resolutions are detected. Then, this refinement selects the simplest resolution and removes the others. We consider the simplest resolution as the one that involves the minimum number of change actions. Finally, in order to guarantee the invariant eliminating negative behavior without altering positive behavior, the condition and guard of the simplest resolution is modified to include the constraints (conditions and guards) of the removed redundant resolutions. R1 implies 4 actions whereas R2+R1 implies 8 actions. Therefore, the refactoring eliminates R2+R1 and modifies the guard and condition of R1 in order to include the constraints of the resolution removed and not lose reconfiguration capacity. Table 5 shows the resolution modified by the refinement:

RECONFIGURATION	CONDITION	GUARD
R1	***cond1 v (cond2^cond1)***	***[(!C8 ^ !C12) v C4 v C10]***

Table 5. Resolution modified to avoid Redundancy in the DSPL.

The possibility space after applying the Redundancy Avoidance refinement is shown in Figure 6.

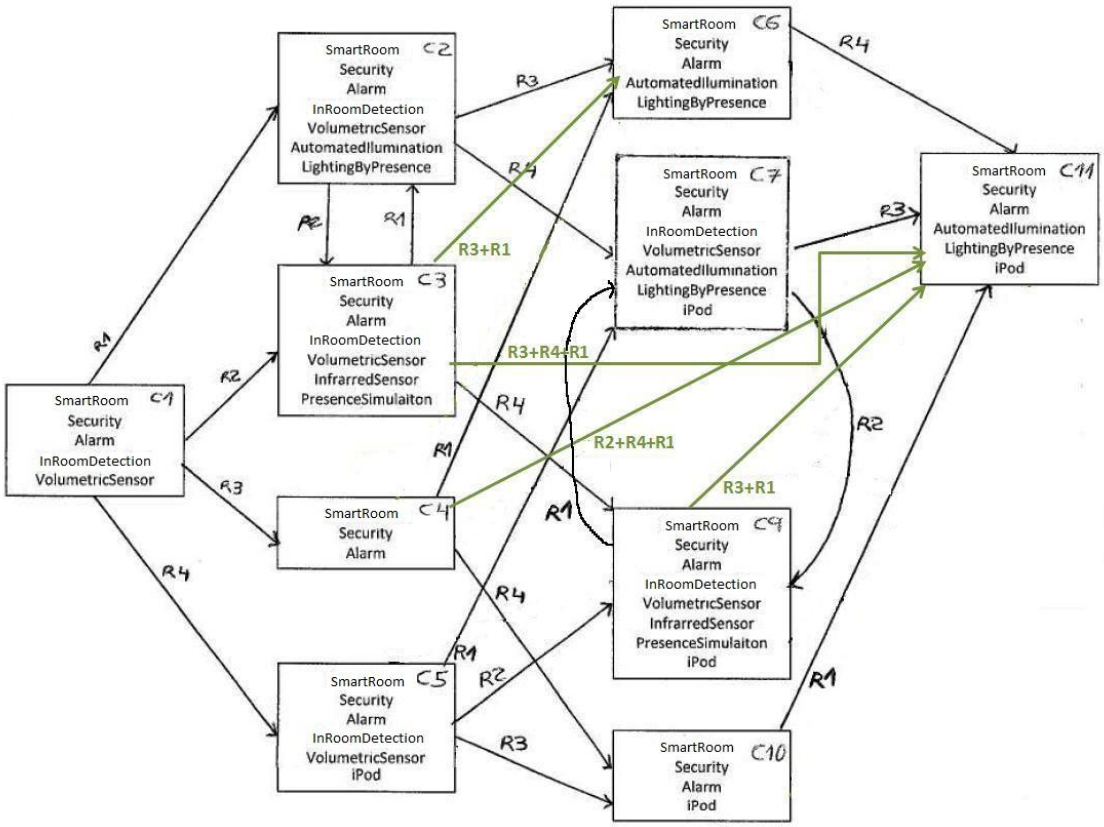

Figure 6. Possibility Space after applying the Redundancy Avoidance Refinement.

c) Property 3: Reversibility

Given the possibility space of Figure 6, it can be observed that the space contains reversible reconfigurations between configurations C2-C3 and C7-C9. These reconfigurations are triggered by resolutions R2 and R1. Table 6 summarizes the reversible reconfigurations triggered by resolutions R2 and R1.

RESOLUTION	SOURCE	DESTINATION
R2	C2	C3
R1	C3	C2
R2	C7	C9
R1	C9	C7

Table 6. Reversible Reconfigurations

Depending on the type of reversible system that is required, either the designer can apply one of the following refinements making either a Total Reversible System or a Nonreversible System.

- **Refactoring 1: Nonreversible system**

The purpose of this refinement is to ensure that there is no reconfiguration that leads directly to the previous configuration. For each pair of reversible reconfigurations, the refinement removes one of the two reconfigurations. In order to not lose reconfiguration capacity, those resolutions are not actually removed. The refinement modifies their guards to constrain when those reconfigurations can be executed.

If the designer requires a Nonreversible System, the Nonreversible System refinement can be applied to achieve it. Then the refinement removes resolution R1 or R2, the designer can choose which of these two resolutions should be removed. Consequently, two different possibility spaces can be obtained:

– If the designer decides to remove reconfiguration R1, the refinement modifies the guard of the resolution R1 to be avoided from the specification. This guard avoids the application of resolution R1 also when the system is on configuration C3 or C9.

RECONFIGURATION	CONDITION	GUARD
R1	cond1 v (cond2^cond1)	[(!C8 ^ !C12 ^ **!C3** ^ **!C9**) v C4 v C10]

Table 7. Resolution R1 after applying the Nonreversible System refactoring removing R1.

– If the designer decides to remove reconfiguration R2, the refinement modifies the guard of the resolution R2 to be avoided from the specification. This guard avoids the application of resolution R2 also when the system is on configuration C2 or C7.

RECONFIGURATION	CONDITION	GUARD
R2	cond2	[!C4 ^ !C6 ^ !C10 ^ !C11 ^ **!C2** ^ **!C7**]

Table 8. Resolution R2 after applying the Nonreversible System refactoring removing R2.

- **Refactoring 2: Total Reversible System**

The purpose of this refinement is to ensure that, for all configurations contained in the possibility space, there exists a reconfiguration that leads directly to the previous configuration. The refinement generates new reconfigurations that assure that every reconfiguration has a reverse reconfiguration. The new reconfigurations define compensation actions to reverse a reconfiguration. The refinement also introduces guards to the new generated resolutions to avoid the generation of new configurations because of the application of this resolution from other configurations. Table 9 illustrates the new resolutions generated after applying the Total Reversible Refinement to the possibility space above (Figure 6). Figure 7 illustrates the possibility space after applying the Total Reversible refinement. The new generated reconfigurations are represented with dashed lines.

RESOLUTION	GUARDA	FEATURES	STATE
$\overline{R3}$	[C6 v C10 v C11]	InRoomDetection	True
		VolumetricSensor	True
$\overline{R4}$	[C5 v C7 v C9 v C10 v C11]	iPod	False
$\overline{R3 + R1}$	[C6 v C11]	InRoomDetection	True
		VolumetricSensor	True
		Automated Ilumination	False
		InfraredSensor	True
		LightingByPresence	False
		PresenceSimulation	True
$\overline{R3 + R4 + R1}$	[C11]	InRoomDetection	True
		VolumetricSensor	True
		iPod	False
		Automated Ilumination	False
		InfraredSensor	True
		LightingByPresence	False
		PresenceSimulation	True
$\overline{R4 + R2 + R1}$	[C11]	iPod	False
		Automated Ilumination	True
		InfraredSensor	False
		LightingByPresence	True
		PresenceSimulation	False
		Automated Ilumination	False
		InfraredSensor	True
		LightingByPresence	False
		PresenceSimulation	True
$\overline{R1}$	[C2 v C6 v C11]	Automated Ilumination	False
		LightingByPresence	False
$\overline{R2}$	[C3 v C9]	InfraredSensor	False
		PresenceSimulation	False

Table 9. New resolutions generated by the refinement.

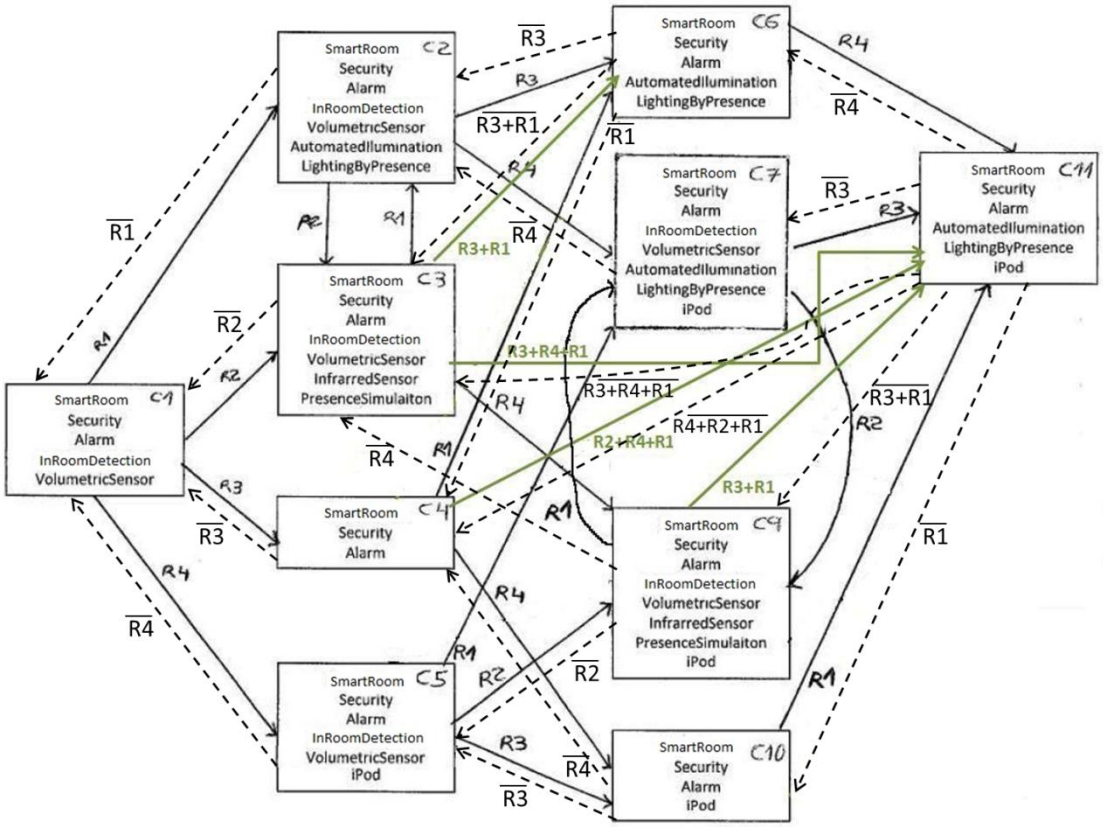

Figure 7. Possibility Space after applying Total Reversible refactoring.

d) Property 4: Contextual Consistency

This property guarantees that when a determined context condition is fulfilled, the system evolves (independently from its current state) to a predefined or controlled configuration. For example, in the Smart Hotel when it is detected fire, the system must always reconfigure to a configuration "Emergency" independently of the current system configuration. In the new configuration "Emergency" (C13) are active the features: *SmartRoom*, *Security*, *Alarm*, *Siren* and *BlinkingLight*.

- **Refactoring 1: Contextual Consistency**

The refactoring generates from each configuration in the possibility space a new reconfiguration that evolves the system to a new configuration "Emergency". Figure 8 illustrates the possibility space after applying this refinement. The new generated reconfigurations are represented with dashed lines.

The refactoring generates the following resolutions:

RESOLUTION	CONDITION	GUARD	FEATURES	STATE
$R_{C1\text{-}C13}$	$cond_{fire}$	[C1]	InRoomDetection	False
			VolumetricSensor	False
			Siren	True
			BlinkingLight	True
$R_{C2\text{-}C13}$	$cond_{fire}$	[C2]	InRoomDetection	False
			VolumetricSensor	False
			Automated Ilumination	False
			LightingByPresence	False
			Siren	True
			BlinkingLight	True
$R_{C3\text{-}C13}$	$cond_{fire}$	[C3]	InRoomDetection	False
			VolumetricSensor	False
			InfrarredSensor	False
			PresenceSimulation	False
			Siren	True
			BlinkingLight	True
$R_{C4\text{-}C13}$	$cond_{fire}$	[C4]	Siren	True
			BlinkingLight	True
$R_{C6\text{-}C13}$	$cond_{fire}$	[C6]	Automated Ilumination	False
			LightingByPresence	False
			Siren	True
			BlinkingLight	True
$R_{C11\text{-}C13}$	$cond_{fire}$	[C11]	Automated Ilumination	False
			LightingByPresence	False
			iPod	False
			Siren	True
			BlinkingLight	True

$R_{C7\text{-}C13}$	$cond_{fire}$	[C7]	InRoomDetection	False
			VolumetricSensor	False
			Automated Ilumination	False
			LightingByPresence	False
			iPod	False
			Siren	True
			BlinkingLight	True
$R_{C9\text{-}C13}$	$cond_{fire}$	[C9]	InRoomDetection	False
			VolumetricSensor	False
			InfrarredSensor	False
			PresenceSimulation	False
			iPod	False
			Siren	True
			BlinkingLight	True
$R_{C10\text{-}C13}$	$cond_{fire}$	[C10]	iPod	False
			Siren	True
			BlinkingLight	True
$R_{C5\text{-}C13}$	$cond_{fire}$	[C5]	InRoomDetection	False
			VolumetricSensor	False
			iPod	False
			Siren	True
			BlinkingLight	True

Table 10. Generated Resolutions after applying Contextual Consistency refactoring.

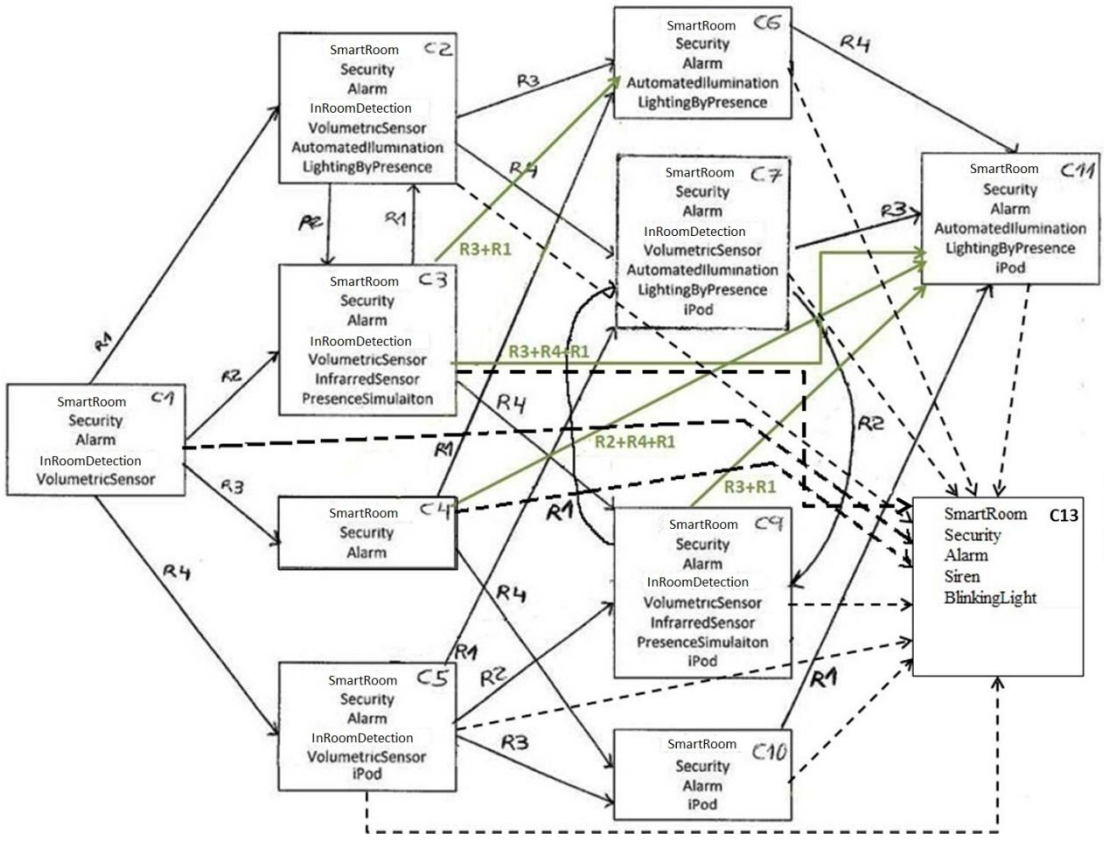

Figure 8. Possibility Space after applying Contextual Consistency refactoring.

e) Property 5: Contextual Determinism

This property involves predictability in the reconfiguration process to future states. That is, for each possible context condition every state in the system has exactly one reconfiguration that leads to the next state. Thus, from a given state, when a determined context condition is fulfilled, the system can only reconfigure to one destination state.

It can be observed that Smart Hotel DSPL defines a deterministic system because each resolution has associated a different context condition. Let's suppose that the designer defines the following resolution:

RESOLUTION	CONDITION	DESCRIPTION	FEATURES	STATE
R5	**cond4**	A person starts working	Laptop	True

Table 11. New defined resolution.

With this new defined resolution, the DSPL defines a nondeterministic system, because the fulfillment of the context condition cond4 can lead to more than one reconfiguration (triggered by R4 or R5) and, therefore, to more than one destination state. It can be applied a refactoring in order to guarantee deterministic systems.

- **Refactoring 1: Contextual Determinism refactoring**

The refactoring associated with this property offers two options to achieve a deterministic system:

- Allows designer to choose the resolution that should be triggered when the context condition is fulfilled.

- Modifies the condition of some resolutions in order to avoid the simultaneous reconfigurations.

References

[1] P. Trinidad, D. Benavides, A. Ruiz-Cortes, S. Segura, and A. Jimenez, “Fama framework,” in Software Product Line Conference, 2008. SPLC ’08. 12th International, 2008, p. 359.