A Software Infrastructure for Executing Adaptive Daily Routines in Smart Automation Environments

Estefanía Serral
Christian Doppler Laboratory
for Software Engineering Integration (CDL)
Vienna University of Technology
Email:estefania.serral@tuwien.ac.at

Pedro Valderas and Vicente Pelechano
ProS Research Center
Universitat Politècnica de València
Email:{pvalderas, pele}@pros.upv.es

***Abstract*—Since the advent of Pervasive Computing, the execution of user daily routines in an adaptive way has been a widely pursued challenge. Its achievement would not only reduce the tasks that users must perform every day, but it would also perform them in a more convenient way while optimizing natural resource consumption. In this work, we meet this challenge by providing a software infrastructure. It allows users' routines to be automated in a non-intrusive way by taking into account users' automation desires and demands. We demonstrate this by performing a case-study based evaluation.**

***Index Terms*—adaptive routine automation; models at runtime;**

I. Introduction

In recent decades, computers have become more and more common in many items such as ovens, refrigerators, coffee makers, mobile phones, tablets, etc. This proliferation of technology brings the building of smart environments closer to becoming a reality. Smart environments provide services to control the items that are used in our daily activities [1]. For instance, there are pervasive services for controlling lights, air conditioner and heating, windows, coffee makers, etc.

One of the final goals of developing smart environments is to automate user daily routines by using these services. A routine is a set of tasks characterized by habitual repetition in similar contexts. For instance, a typical routine could be the following. Every working day at 8 o'clock, Bob's alarm clock goes off. Bob wakes up, switches the lights on and stops the alarm. Then, to take a shower, Bob turns on the bathroom heating if it is cold. Finally, after getting dressed, Bob goes to the kitchen and makes a coffee for breakfast.

Due to the fact that people are creatures of habit, we perform numerous daily routines such as the one presented above. Several works, such as [2][3][4] [5], have dealt with performing these routines on the users' behalf; however, their solutions may lead to intrusive systems that automate tasks that users do not necessarily want automated. In this work, we present a software infrastructure capable of automating users routines in a non-intrusive way. Due to the complexity of human behaviour, user participation is necessary in order to avoid intrusiveness when attempting to fulfill user demands. For this reason, the infrastructure that we propose makes use of models during runtime. These models allow routines to be represented by using high-level concepts that are close to user knowledge. This helps users to understand the routines to be automated and to participate in their design. By simply interpreting the models at runtime, the infrastructure can automate the routines as described.

With this infrastructure, we could make users' lives easier and provide them with a higher quality of life: they would not have to waste their time or worry about the tasks that could be automated (e.g., Bob will never oversleep in the morning because the alarm clock is set automatically). In addition, these tasks could be performed more efficiently and in a more convenient way for users since tasks can be analyzed before being automated using the models (e.g., heating could be turned on 10 minutes early so that the bathroom is already hot when Bob takes a shower; instead of the alarm clock going off, Bob's preferred radio channel could be used). Moreover, routines could self-adapt according to context (e.g., blinds could be raised if it is a sunny day instead of switching lights on) and could help to reduce natural resource consumption by applying the advice provided by experts on controlling lighting, heating and air conditioning, taps, and so on (e.g., all lights could be automatically switched off when the inhabitants leave home; blinds could be lowered in summer when nobody is at home so that it is not so hot when the inhabitants arrive).

The rest of the paper is organized as follows. Section II describes the related work. Section III explains essential requirements for routine automation. Section IV presents the software infrastructure that automates routines in a context-adaptive way. Section V validates the approach using a case study based evaluation. Section VI concludes the paper.

II. Related Work

Related work can be subdivided into machine-learning approaches, context-aware rule-based approaches, end-user centered approaches, and task-oriented computing.

Machine-learning approaches have attempted to deal with the automation of user routines by automatically inferring them from past user actions [2][3]. These approaches have done excellent work by automatically learning from user behaviour; however, they have some drawbacks. They may be intrusive for users because they do not usually take into account users' desires (e.g., the repeated execution of an action does not imply that the user wants this automation). Also, they

 ISBN: 978-1-61208-274-5

reproduce the actions that users have frequently executed in the past and in the same manner that they were executed. This prevents user tasks from being carried out in a more efficient and convenient way and does not allow tasks to be automated if they were not performed by users (e.g., closing windows when users are not at home and it starts to rain).

Context-aware rule-based approaches have made great advances in introducing context into software systems. To automate user tasks, they program rules that trigger the sequential execution of actions when a certain context event is produced [4] [5]. However, although context information is taken into account, these works do not usually consider the personal desires of each user; therefore, they may still be annoying. Furthermore,these techniques are only appropriate for automating relatively simple tasks [6]; hence, they usually require large numbers of rules. If we also consider that these rules have to be manually programmed [6], the understanding and maintenance of the system may become very difficult.

End-user centered approaches provide alternatives for end-users to program their environments [7][8]. Most of these approaches are focused on end-user programming by presenting particular UIs and languages. These approaches generally provide better user control. However, they have limited capacities to help end-users build the automations. Therefore, they are only appropriate for developing simple tasks commonly described in the literature, such as controlling lights or doorbells.

Task-oriented computing uses task modelling to facilitate the interaction of users with the system. These systems have proven that task modelling is effective in several fields such as user interface modelling [9], assisting end-users in the execution of tasks [10], etc. These works show the growing usage of task modelling and its remarkable results and possibilities to model system behaviour. However, none of these works attempt to automate adaptive daily routines. Hence, they neither provide enough expressiveness to specify adaptive routines nor enough accuracy to allow their subsequent automation.

III. Requirements for Routine Automation

The users' tasks automation is a delicate matter. The execution of an undesired task will be intrusive for users, and may bother them, interfere in their goals, or even be dangerous; all of which would eventually cause the loss of user acceptance of the system. For instance, consider that the outside door and the security system have been programmed so that the door is automatically locked and the security system is automatically activated when the inhabitants leave home. This can be useful because they will not have to do these tasks anymore, but it can also be a burden if the inhabitants are absent-minded: they will have to unlock the outside door and deactivate the alarm every time they forget something. To prevent these intrusive situations, the following aspects are required:

- **The routines must be automated according to the users' desires and demands**. This is essential so that the routines to be automated are those that users want and are automated the way they want them to be. Due to the technical context, and the imprecise and ambiguous nature of human behaviour, it is very difficult for a system to sense or infer this information. Therefore, **the participation of the users** is necessary in order to fulfill their automation desires and demands [11].
- **The routines must be adaptive to context**. Context information is essential to be able to execute the routines in the opportune situation. For instance, in the routine used as the example, it would be intrusive if the bathroom heating is switched on when the temperature is high or if the radio is turned on anytime. Therefore, routines must be described in a context-adaptive way (e.g., the bathroom heating must be automatically switched on at 7:50 on working days only if the temperature is low, and the radio must be turned on 10 minutes later).
- **Routine adaptation must be facilitated at runtime**. Some routines might never change in user life; however, most of them will. Users' behaviour usually changes over time and the automated routines need to be adapted to these changes. Otherwise, the system may become useless and intrusive. Since these types of changes cannot be anticipated at design time, the automation of routines must be performed in such a way that their adaptation after system deployment is facilitated at runtime.

IV. The Software Infrastructure

A smart environment is developed to provide pervasive services that serve people in their everyday life. These services are in charge of interacting with physical devices in order to change the state of the environment and to sense context. On top of these services, we develop a software infrastructure (see Figure 1) that is designed to properly automate user routines satisfying the requirements explained in the previous section:

- To **facilitate user participation**, the software infrastructure makes use of design models at runtime. It provides a task model that describes the routines by using concepts of a high-level of abstraction that are close to the domain and to user knowledge (concepts such as task, preference, location, etc.). This helps end-users to participate in the routine description since it allows them to focus on the main concepts (the abstractions) without being confused by low-level details [12].
- To execute the described routines **in a context-adaptive way**, the task model describes each routine as a coordination of pervasive services that are performed in the opportune context, i.e., in a context-adaptive way. In addition, in order to be aware of the current context and to be able to automate the routines accordingly, the software infrastructure provides a context manager. It dynamically manages the context changes produced at runtime by using a context repository.
- To automate the described routines in such a way that **their adaptation after system deployment is facilitated**, the software infrastructure has an automation engine that directly interprets the task model at runtime. The model is machine-processable and precise enough to provide the infrastructure with all the information needed to execute

 ISBN: 978-1-61208-274-5

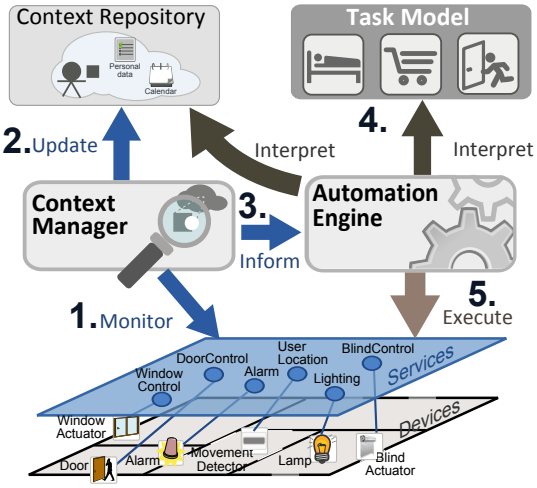

Fig. 1. Runtime infrastructure

the routines. Therefore, when a context change is detected by the context manager, it informs the engine. The engine then reads the routine information from the task model and executes the corresponding pervasive services according to context. With this strategy, the task model is the only representation of the routines to be automated. This allows the routines to be adapted by simply updating the model. As soon as it is changed to adapt the routines, the changes are also taken into account by the engine.

Thus, the infrastructure provides the following main components (see Figure 1): the *context-adaptive task model*, which describes each routine as a context-adaptive coordination of pervasive services; the *context manager*, which is in charge of monitoring context changes at runtime, updating the context repository accordingly, and informing the automation engine about the changes; and the *automation engine*, which is in charge of automating the routines in the opportune context by interpreting the models.

A. The Software Infrastructure in Execution

The software infrastructure executes the routines as described in the task model. This model allows the routines to be described precisely and at a high level of abstraction. As an example, Figure 2 shows the modelling of the routine used in the introduction (the *WakingUp* routine). The root task of the hierarchy represents the routine and is associated to a context situation, which indicates the context conditions whose fulfilment starts the execution of the routine (*WorkingDay=true AND CurrentTime=7:50*). The root task is broken down into simpler tasks (*turn on bathroom heating*, *turn on the radio*, *illuminate the room* and *make coffee*). An intermediate task must be broken down until the leaf tasks can be executed by an available pervasive service. Each leaf task must be related to a pervasive service that can carry out the task. For instance, the *turn on the radio* task is associated to a pervasive service that interacts with the radio to turn it on. This relation is established by simply indicating the service identifier.

If the tasks of the same parent are related to each other, they are carried out in a sequential order according to the indicated temporal relationships. These relationships may depend on context. Thus, in the example, the heating is turned on first; ten minutes later, the radio is turned on and the room is illuminated; and finally, a coffee is made when the user enters the kitchen.

A task can have a context precondition (represented between brackets), which defines the context conditions that must be fulfilled so that the task is performed (e.g., the *turn on bathroom heating* task is only executed if the temperature is low). If the tasks of the same parent are not related to each other, only the first task whose context precondition is satisfied is executed; e.g., to illuminate the room, if the outside brightness is low, lights will be switched on; otherwise, blinds will be raised.

To automate the routines as described in the task model, the software infrastructure performs the following steps (see Figure 3):

1) **Detecting context changes:** A context change is physically detected by a sensor, which is controlled by a pervasive service in the smart environment. The context manager monitors all these services to check context changes. For instance, the context manager monitors the Clock service to detect the time changes. When a change is detected (e.g., it is 7:50 a.m.), the manager updates the current context in the context repository and notifies the engine about this change.
2) **Checking context situations:** After receiving the notification of a context change, the engine analyzes the context situations of the routines specified in the Task Model to check if any of them depend on the context change. Then, by making use of the context manager, the engine checks if any of those context situations are

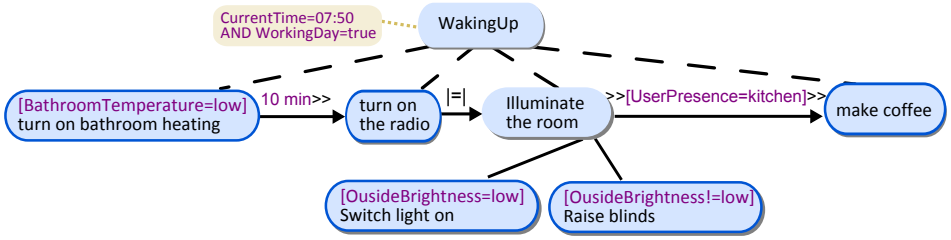

Fig. 2. Routine task for waking up the user

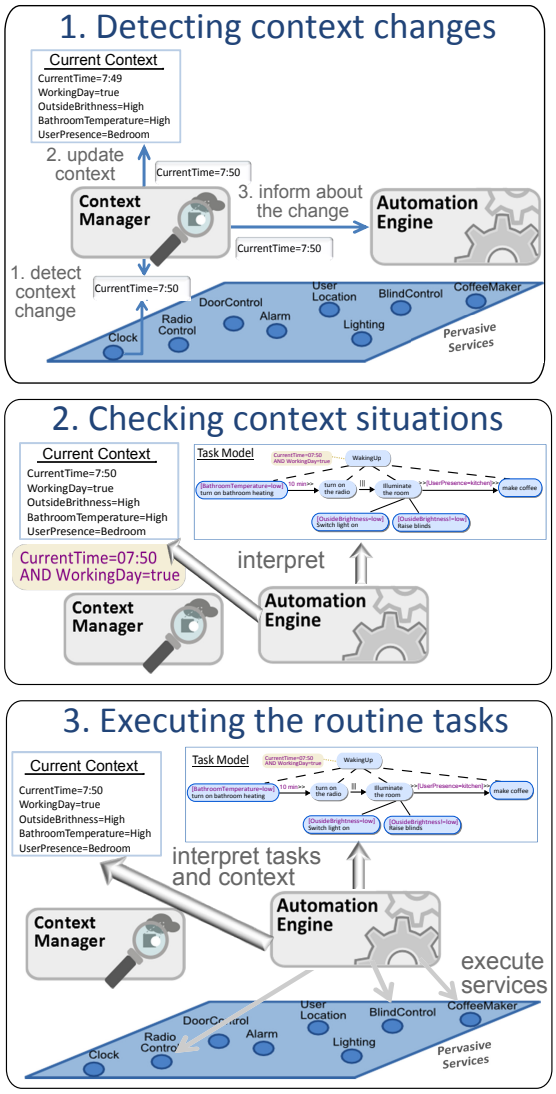

Fig. 3. A possible execution of the WakingUp routine

fulfilled. For instance, when the context manager notifies the engine that it is 7:50 a.m., the engine gets the context situations that depend on time, such as the one for the WakingUp routine, and checks them. On a working day, the engine checks that the context situation of the the WakingUp routine is satisfied.

3) **Executing the routine tasks:** The engine executes the routines whose context situation is satisfied. The engine uses the context manager to check the context conditions. To execute each routine, the engine executes its leaf tasks according to their refinements, their context conditions in the current context, and their temporal relationships. For instance, to automate the WakingUp routine, the engine gets the first subtask (*turn on bathroom heating*) and checks its precondition (*BathroomTemperature=low*). If it is true, the engine executes its related service. The engine then waits 10 minutes, as its relationship with the next task indicates. After that, the engine executes the service related to the *turn on the radio* task. The engine then gets the next task, which is the *illuminate the room* task. To execute it, the engine gets its first subtask (*switch light off*) and checks its context precondition (*OutsideBrightness=low*). If it is satisfied, the engine executes the service related to the task for switching lights on. Otherwise, the engine executes the service related to the *raise blinds* task because its context precondition is the opposite one (*OutsideBrightness!=low*).

Finally, the engine gets the last task. This is related to the previous task by the >>[UserPresence=Kitchen]>> relationship; therefore, the engine waits until Bob enters in the kitchen and then executes the makeCoffee service.

B. Implementation Details

To describe the task model, we have developed a graphical editor using the Eclipse platform, and the EMF and GMF plugins. By using this editor, the model can be graphically edited as shown in Figure 2. These descriptions are stored in XMI (XML Metadata Interchange), which is machine-interpretable at runtime. The context repository is represented as an OWL (Web Ontology Language) ontological model. OWL is an ontology markup language W3C standard that greatly facilitates runtime interpretation and reasoning.

The *context manager* and the *automation engine* are implemented in Java/OSGi technology and are run in an OSGi server together with the pervasive services. Note that the infrastructure is decoupled from the service implementation since we only need to indicate a service identifier.

Using OSGi, the context manager can listen to the changes produced in the services to detect context changes and can also inform the engine when a change is detected. To execute a task, the engine searches for the pervasive service associated to the task in the OSGi server by using its service registry. Then, the engine executes the corresponding service by using the Java Reflection capabilities.

To manage the task model at runtime, the engine uses the EMF Model Query plugin that allows a system to work with any model by querying its structure at runtime. To manage the context repository at runtime, the context manager uses the OWL API 2.1.1, which provides facilities for creating, examining, and modifying an OWL model; and the Pellet reasoner 1.5.2., which allows the OWL model to be queried.

More technical details can be found in [13].

V. Validation of the Proposal

In order to validate the presented software infrastructure, we have applied a case-study based evaluation by following the research methodology practices provided in [14].

The purpose of the evaluation was to validate that our software infrastructure supports the execution of adaptive routines and only automates the routines that users want and in the way they want them. To validate this, we evaluated the following research questions according to the requirements presented in Section III:

 ISBN: 978-1-61208-274-5

1) Does the infrastructure facilitate user participation to take into account user automation desires and demands?
2) Does the infrastructure correctly automate the routines in a context-adaptive way?
3) Does the infrastructure allow routines to be adapted by changing the task model at runtime?

We now summarize the results of this evaluation. More details can be found in [13]. Also, we have recorded several videos that show the software infrastructure in execution. They can be found at http://www.pros.upv.es/art.

A. User Participation

We designed and developed 14 case studies in the smart home domain, covering different set of inhabitants (families, couples and single people). We selected smart homes because this a is fertile ground for offering products and services to improve people' lives. Specifically, the overall purpose of the developed case studies was to make inhabitants' lives more efficient and comfortable and to save energy consumption. A total of 18 subjects between 26 and 57 years old participated as the clients of the case studies (8 female and 10 male). Ten of them had a strong background in computer science, while the rest only had basic computer knowledge.

We identified from 6 to 12 routines to be automated in each case study resulting in a total of 97 routines. It took us between 40 and 90 minutes to specify the routines of each case study using the task model. We then briefly taught the subjects about the main components of the task model notation and evaluated their comprehension to determine if the task model facilitated user participation. To do this, we used a short semi-structured interview in which we asked the subjects questions to make them reason about the task model. For instance, some of these questions were: how many tasks will be executed in this routine?; when will this routine be activated?; when will this task be executed?.

We found that 14 of the 18 subjects understood the routines specified in the task model perfectly. The other 4 users, those with little mathematics and computer skills, understood the structure of the model (task hierarchy and task relationships) very well; however, they had difficulty knowing what the used context conditions meant. To solve this problem, we added a new view in the task model editor to show these conditions in natural language. For instance, instead of showing *[OutsideBrightness=low] switch lights on* in the model, we show: *if the outside brightness is low, switch lights on*, or instead of showing $>> [UserPresence = Kitchen] >>$, we show *when you arrive to the kitchen*.

After checking the subjects' comprehension of the model, we explained the specified routines to them. We found that the task model is very useful in discussing and validating the routines to be automated. If something was not specified the way the users wanted it to be automated, we refined the model to fulfil their requirements. We repeated this process until the users agreed with the specification. This allowed us to describe the routines by taking into account the automation desires and demands of the users.

B. Context-Adaptive Routine Automation

Once the task models were validated, we put the system into operation to automate the described routines. We used a scale environment with real devices (see http://pros.upv.es/art) to represent the Smart Home. This execution environment was made up of a PC and a network of KNX devices connected to the PC by a USB port. An Equinox distribution (which is the OSGi implementation of Eclipse) was run in the PC. The software infrastructure together with the pervasive services required to execute the leaf tasks of the routines (a total of 26 services) were installed and started in Equinox.

By using JUnit tests, we validated that the routines were correcly automated in a context-adaptive way. Specifically, the following aspects were validated:

- All the routines were triggered only when its context situation was fulfilled.
- When a routine was executed, all the required services were executed in the correct order and in the correct context conditions.

The proposed validation consisted in: (1) simulating the fulfilment of specific context conditions in order to trigger the execution of several routines, and (2) checking that all the services that must be executed were registered by the context manager in the correct order, respecting the corresponding temporal relationships between the tasks.

We performed this process in an iterative way, which allowed us to detect and solve some mistakes. For instance, we realized that the routines dependent on time, made the system enter into a loop. This was because the system updated time every second and the smallest time unit considered in the routines was minutes. Thus, the context situation of these routines was continuously fulfilled until a minute went by. To solve this problem without overloading the system, we updated the context manager to update time every minute.

C. Routine Adaptation after System Deployment

We validated that the routines could be easily evolved by changing the task model at runtime. Specifically, we changed the task model to perform the following types of adaptation: delete routines; modify routines by changing their context situation and their tasks (task order, context preconditions, temporal relationships, etc.); and add new routines.

After each adaptation, we simulated the fulfilment of the context situations of the routines and applied the JUnit tests again to check that the routines were correctly executed according to the performed evolution. For instance, we modified the *WakingUp* routine. We changed the second task in order to wake Bob up with relaxing music; we removed the lighting task, and added a new task so that the system informed Bob about the weather when he was in the kitchen. Figure 4 shows these modifications in the task model and the execution trace of the *WakingUp* routine before and after evolving it.

VI. Conclusion

In this work, we have presented and evaluated a sofware infrastructure that achieves the automation of adaptive daily

 ISBN: 978-1-61208-274-5

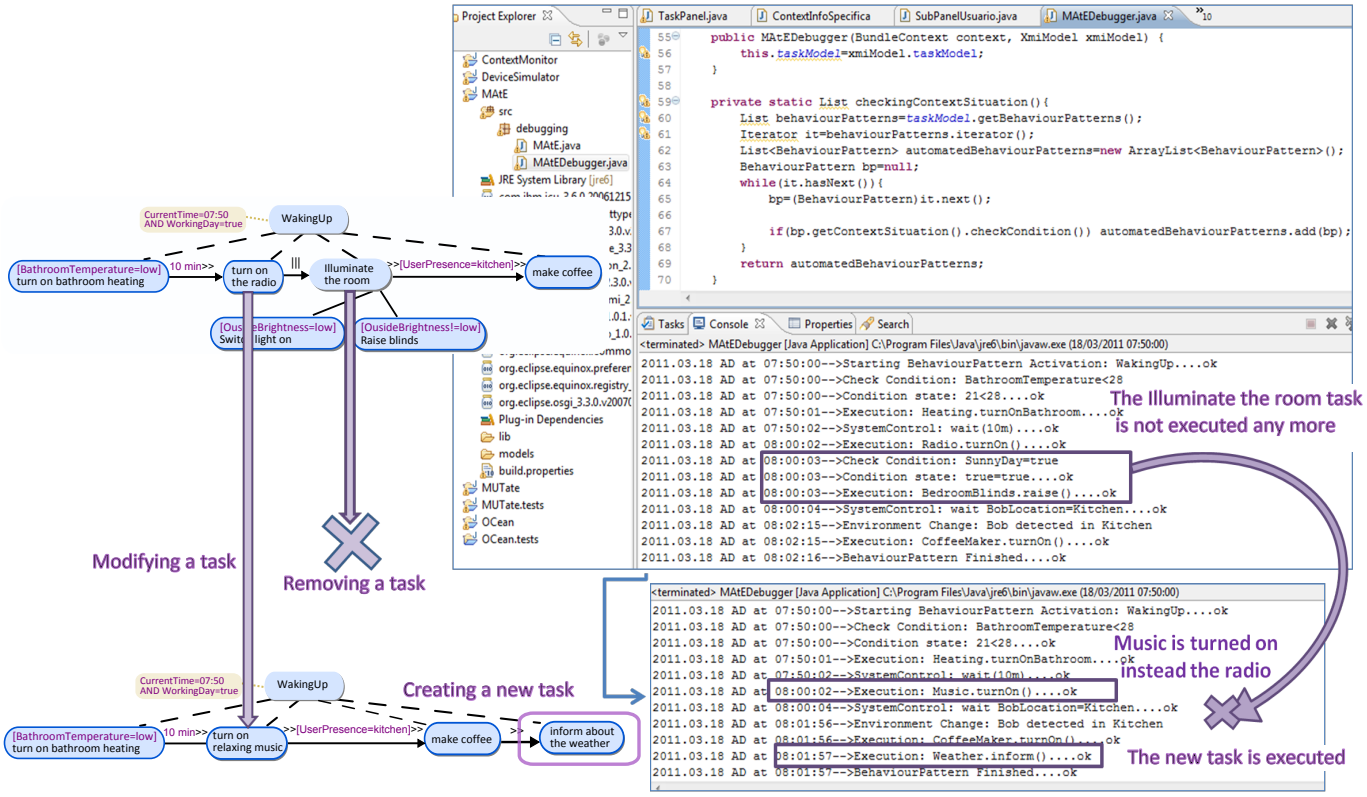

Fig. 4. Execution traces before and after evolving the WakingUp routine

routines. These routines are represented in high-level abstraction context-adaptive models that are directly interpreted at runtime. This considerably facilitates the further adaptation of the routines by changing the models (i.e., at the modelling level) at runtime, which is one of the top challenges in software evolution research [15]. As soon as the models are changed to adapt the routines, the changes are also taken into account by the automation engine.

Further work will be dedicated to extending the approach with machine-learning algorithms in order to provide more automation in the requirements capture and the routine adaptation after system deployment. When the system is running, the context manager stores the user actions in the context repository. Machine-learning algorithms can use this information to detect new routines or changes in the ones already specified and adapt the models accordingly.

ACKNOWLEDGMENT

This work has been supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria.

REFERENCES

[1] F. Mattern, "The vision and technical foundations of ubiquitous computing," *Upgrade European Online Magazine*, pp. 5–8, 2001.

[2] H. Hagras, V. Callaghan, M. Colley, G. Clarke, A. Pounds-Cornish, and H. Duman, "Creating an ambient-intelligence environment using embedded agents," *IEEE Intelligent Systems*, vol. 19, no. 6, pp. 12–20, 2004.

[3] P. Rashidi and D. J. Cook, "Keeping the intelligent environment resident in the loop," in *IE 08*, 2008, pp. 1–9.

[4] K. Henricksen, J. Indulska, and A. Rakotonirainy, "Using context and preferences to implement self-adapting pervasive computing applications," *Software: Practice and Experience*, vol. 36, no. 11-12, pp. 1307–1330, 2006.

[5] M. García-Herranz, P. Haya, and X. Alamán, "Towards a ubiquitous end-user programming system for smart spaces," *Journal of Universal Computer Science*, vol. 16, no. 12, pp. 1633–1649, 2010.

[6] D. Cook and S. Das, *Smart environments: Technology, protocols and applications*. Wiley-Interscience, 2004, vol. 43.

[7] A. K. Dey, R. Hamid, C. Beckmann, I. Li, and D. Hsu, "a cappella: Programming by demonstration of context-aware applications," *CHI 2004*, pp. 33–40, 2004.

[8] J. Chin, V. Callaghan, and G. Clarke, "A programming-by-example approach to customising digital homes," in *IE 08*, 2008, pp. 1–8.

[9] C. Pribeanu, Q. Limbourg, and J. Vanderdonckt, "Task modelling for context-sensitive user interfaces," *Interactive Systems: Design, Specification, and Verification*, pp. 49–68, 2001.

[10] R. Huang, Q. Cao, J. Zhou, D. Sun, and Q. Su, "Context-aware active task discovery for pervasive computing," in *International Conference on Computer Science and Software Engineering*, 2008, pp. 463–466.

[11] F. M. Reyes, "Issues of sensor-based information systems to support parenting in pervasive settings: A case study," *Emerging Pervasive and Ubiquitous Aspects of Information Systems: Cross-Disciplinary Advancements*, p. 261, 2011.

[12] F. Paternò, "From model-based to natural development," *HCI International*, pp. 592–596, 2003.

[13] E. Serral, "Automating routine tasks in smart environments. a context-aware model-driven approach," Ph.D. dissertation, Technical University of Valencia, DSIC, 2011.

[14] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[15] T. Mens, "The ercim working group on software evolution: the past and the future," in *IWPSE-Evol workshops*. ACM, 2009, pp. 1–4.

 ISBN: 978-1-61208-274-5