TOWARDS CONTEXTUAL TASK PATTERNS FOR SMART MEETING ROOMS

Michael Zaki, Peter Forbrig and Jens Brüning
Department of Computer Science, Rostock University, Albert-Einstein Str. 22, Rostock, Germany

Keywords: HCI, Smart Environment, Task Model, Context, Task Pattern, Precondition, Effect.

Abstract: The main idea of smart environments is to deliver proper assistance to the resident users while performing their daily life tasks. Thus, task models are convenient as a starting point for developing applications for those environments, as they give the developer the opportunity to focus on the users and their tasks. In such an environment, mutual dependencies between different types of entities are taking place and affecting the way the user is executing the tasks. Therefore, other models (e.g. device model, location model …etc.) have to be developed and linked to the task model in order to truly illustrate how the tasks are executed in those environments. Due to the increasing number of models and the dependencies among them, modeling an interactive application to be operated in such an environment is a tedious and overwhelming process. In this paper, we present an attempt to overcome the modeling complexity by fostering the concept of reuse on a high level of abstraction using task patterns. We extend the former definition of task patterns by integrating the environmental preconditions and effects within the pattern structure in order to maximise the benefit of the usage of those patterns.

1 INTRODUCTION AND RELATED WORK

In (Cook, 2004) a smart environment is defined as "*a small world where different kinds of smart devices are continuously working to make inhabitants' lives more comfortable*". From this definition, one can infer that the main goal behind such environments is to assist the inhabitants in their daily life's tasks in an efficient and implicit manner. However, one of the main preconditions for offering such assistance is to have a clear idea about the user's intentions. Thus, we need to model the user's expected behavior beforehand and then based on this model we can infer how the environment should interact with the user in order to minimize the task performance burden and to let the user have a positive impression about the environment.

Task models have usually been used as a tool to elicit requirements in the early development stages. Nowadays they are playing a more influencing role as an appropriate starting point for interactive processes development. For example, (Feuerstack, 2006) presents an attempt to create a first draft of the user interface based on task trees. Also (Blumendorf, 2010) suggests the usage of dynamic task models in order to build adaptive user interfaces. However, isolated task models cannot express all relevant information for tasks execution in domains with high complexity like smart environments. Other environmental entities and factors (e.g.: devices, objects, user position…etc.) are constraining and affecting the way tasks are performed. Thus, in (Wurdel, 2008) the collaborative task modeling language (CTML) enables the integration of the execution domain within the modeling process. In CTML, a dedicated model is needed for every environmental aspect to be taken into account.

Whereas CTML seems to be suitable as a modeling framework for smart environment applications, building all those models and the identification of the mutual dependencies among them is a real burden for the developer even if we take into consideration the existence of a tool supporting the development of those models. Consequently, in this paper we are trying to overcome the previously described problem by fostering the concept of reuse on a high level of abstraction using task patterns. However, we argue for a broad definition of task patterns in the context of smart environments by integrating the restricting

Zaki M., Forbrig P. and Brüning J..
TOWARDS CONTEXTUAL TASK PATTERNS FOR SMART MEETING ROOMS.
DOI: 10.5220/0003824801620169
In *Proceedings of the 2nd International Conference on Pervasive Embedded Computing and Communication Systems* (PECCS-2012), pages 162-169
ISBN: 978-989-8565-00-6

preconditions for the task execution and the effects resulting due to this execution within the pattern's structure. This way the developer is not only assisted while building her task model, but the integration of the task template provided by the pattern with all its dependencies in the other relevant models (e.g.: device model, object model…etc.) is also feasible. Thus, we believe that those patterns can make the modeling process less time-consuming and of better performance.

Despite the fact that patterns were first invented in urban architecture by Christopher Alexander in 1977 (Alexander, 1979), their influence has spread and reached the software engineering as well as the HCI area. In (Gamma, 1994) the Gang of Four have introduced patterns as recurring solutions for common software design problems. These patterns realized a brilliant success in the software engineering field, and consequently the idea of using patterns has also conquered the HCI domain. (Borchers, 2001) defines HCI design patterns as "*a structured textual and graphical description of a proven solution to a recurring design problem*".

Moreover, the notion of task patterns is not totally new. The idea was initially introduced by (Breedvelt, 1997), where he identified reusable structures or templates and suggested their usage in the application's design process. Afterwards, this idea was extended in (Sinnig, 2004) where the authors suggested using patterns as generic reusable task fragments serving as building blocks for the creation of task models. They also introduced domain variables as placeholders to integrate the context of use. Additionally, (Wurdel, 2007) presents an attempt to adapt task patterns according to the context of use by taking benefit of the concept of "decision nodes" proposed by (Luyten, 2004). Sub-nodes of a given decision node are annotated with contextual constraints to decide when a specific path should be taken.

From our point of view, the already defined task patterns in previous approaches are suitable for developing HCI applications where users are interacting with a designed UI which may run on several devices in various locations. However, when examining the domain of smart environments (e.g.: smart meeting rooms) one can notice that whether a specific task can or cannot be executed at a given time 't' may depend on numerous environmental factors (e.g.: device state, existence of a specific object, actor location…etc.) or in other words the state of the environment at this specific time. Thus, we believe that a seamless integration of those mandatory preconditions within task patterns designed for smart environments is of considerable interest. In other words, the already mentioned attempts for developing task patterns take into consideration the changes occurring to the way the task should be executed under various environmental conditions. Our approach aims to extend this idea by expressing the mandatory preconditions and effects for every individual task within the task template offered by the pattern as a solution. (i.e. :Without those constraints, the task cannot take place).

The paper is structured as follows: In section 2 our new understanding of the term "task pattern" in the context of smart environments and the suggested structure of our patterns are presented. Afterwards, in order to make our ideas more concrete a pattern example is illustrated in section 3. Finally, we summarize our ideas and we give a brief overview of future research avenues in this area.

2 TASK PATTERNS FOR SMART ENVIRONMENTS

As already discussed, various factors may affect the executability of the required tasks in a given smart environment. These factors are formed by the state of the surrounding intervening entities and thus the environmental context in which the task occurs. However, we distinguish here between two types of context. First of all, we identify the so-called "forcing context" as *"The initial state in which the environment should be before the task execution and the final state in which it should be after this execution"*. Secondly, we define the "flexible context" as *"Any information which may change the way the task is executed in the environment or the forcing context in which the task occurs"*. While the first sort is mandatory as a pre-request for the task execution, the second one determines in a precise way how the task is being executed given the current devices and objects used to perform the task. We claim that the forcing context can be adapted according to the flexible one, as the state in which the environment should be before or after the task execution depends on the exact way the task is performed. For example, a user taking notes of a presentation given in the environment will need a device containing a text editor in case this user decides to take notes electronically, while in case she prefers to take notes by handwriting a pen is actually needed.

To make the distinction between the concept of forcing context and flexible context clearer, let us

take the case of a user playing the role "presenter" in a presentation scenario taking place in a smart meeting room. This user cannot give her talk if she does not have in possession any slides to present. Thus, the slides can be considered as one of the elements of the forcing context for the task "Give Talk". Additionally, a smart meeting room may give the user the possibility of using numerous projectors and canvases for one presentation. However, whether the presenter decides to use only one projector or several ones to give her presentation, the talk can be given. That is only a specification of the exact way in which the task is performed. Thus, that is an example of an element in the flexible context to which the task execution is adapted.

From our point of view, the solution provided by a convenient task pattern to be employed as a building block in the user's task model should encapsulate the integration of the forcing context (i.e.: preconditions and effects) in the task template and should also make the template adaptable to the given flexible context. In other words, the task pattern should enable the representation of the required preconditions and effects and should be generic enough to be adaptable to several environmental settings. However, our goal is not to explicitly integrate the forcing context in the task model itself, but to provide it to the developer in a visualized way in the pattern solution so that he can have a comprehensive image about the exact way in which the task should be executed in the environment.

In order to truly express the environmental dependencies within our patterns, the pattern library we are developing should not be defined for the whole domain of smart environments, as the nature of tasks to be executed and the objects and devices in need differ from one specific domain to another one. Therefore, we decided to stick to the domain of smart meeting rooms as a subset of smart environments domain while developing our patterns. Whereas one might think that the fact that the pattern library is not addressing the whole smart environment domain might be a disadvantage, we argue that in order to maximize the benefit gained by those patterns, being bound to a more precise domain is highly advisable. Moreover, we argue that while the patterns themselves are restricted to smart meeting rooms field, but the methodology we adopted in order to extract those patterns (and which will be discussed) can be followed in order to extract patterns in other smart environment areas (e.g.: smart homes, smart offices,…etc.) and thus, pattern libraries addressing those areas can be similarly identified and developed.

A smart meeting room is an example of a collaborative environment, where numerous actors are exchanging information and collaborating together so that they can achieve a final high level common goal. Such a goal can be identified as a team goal. Consequently, every task to be executed by a given user in the environment is in a way or another contributing as a step towards the final team goal (e.g.: Do a presentation) and additionally her own individual goal in the environment (e.g.: Attend a presentation). Thus, as a first step to develop our patterns we started by identifying team goals which may take place in a given smart meeting room. We ended up having the following six main team goals formulized as final states: *"Conference session performed", "lecture given", "work defended", "topic discussed", "debate managed" and "video watched"*.

By investigating the above mentioned goals we can move to the second step which is the extraction of various roles which may be played by the resident actors in the environment to finally achieve those goals. For example, in a conference session scenario the roles *presenter*, *listener* and *chairman* can be easily identified. Once we determine all the included roles in those scenarios, repetitive task templates existing in several task models are collected and used as task patterns. In this way we built our pattern library which is composed of around twenty task patterns. The exact structure of our patterns as well as their adaptability is explained in further details in the next sub-section.

2.1 Task Pattern's Structure and Adaptability

Unlike the former task patterns suggested in (Breedvelt, 1997); (Sinnig, 2004) and (Wurdel, 2007), we aim to integrate the forcing context within the description of the solution provided by our patterns. In Fig.1, we present a meta-model illustrating the relation between the two components of the pattern solution which are: the task template suggested by the pattern and the various environmental dependencies. By having a look at Fig.1, one can infer the different factors constituting the preconditions and effects which constrain the execution of every individual task within the task template. Briefly, the execution of a given task may depend on the state of a stationary or dynamic device, the existence of a certain object, and the properties of the actor performing the task and her position within the environment.

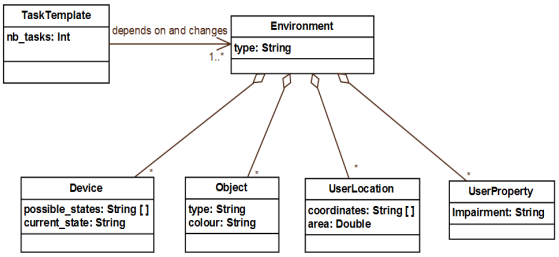


Figure 1: A meta-model presenting the relation between the task template and the surrounding environment.

To have a clear idea about the exact structure of our contextual task patterns, one of the simplest patterns in our library is presented in table 1. Inspired by the early work of (Wurdel, 2007) and (Paterno, 2001), we defined the skeleton of our task patterns. Every pattern is composed of 7 mandatory sections namely, "Id", "Name", "Problem", "Situation", "Solution", "Diagram", "Adaptation variables" and an optional one "Referenced patterns". While a visualization of the defined solution is represented in the *"diagram"* section, the last mandatory attribute *"adaptation variables"* is determining the components of the flexible context to which the pattern is sensitive. Finally, we have the only optional section *"referenced patterns"* listing all the patterns which are referred to within the solution of this pattern. Every task pattern is adapted to at least one context-dependent variable. Taking into account that assisting the user is the first duty of a given smart environment, and as the main trend nowadays is the design of universally accessible applications, we decided to take impaired users into account while developing our patterns. In (Zaki, 2011) the authors propose the usage of the so-called accessibility patterns in order to alter the way a given task is executed and make it suitable for an impaired user depending on the kind of her impairment. For that, they assume the existence of a methodology changing the format of the information to another suitable one for this user. We extend this approach by restructuring those patterns so that their solution can have the same skeleton of our task patterns' solution.

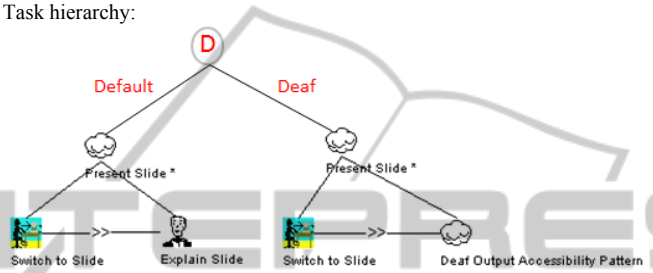
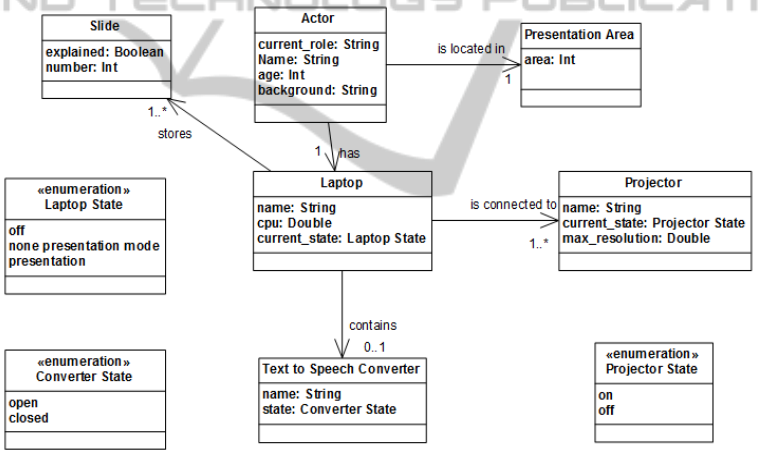
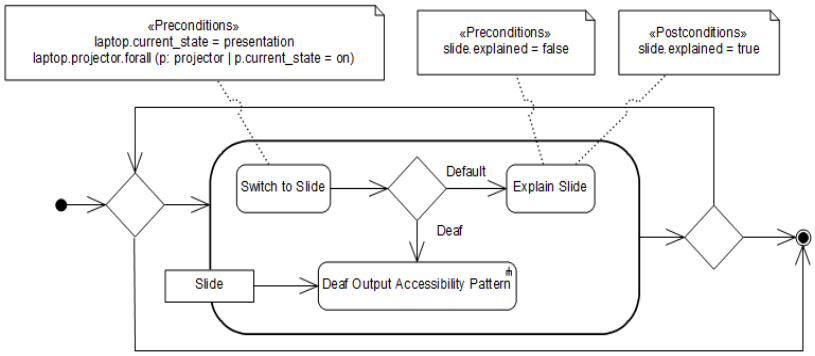
Other than being adapted to impaired users, every task pattern in our library can also be adapted to one or more attributes depending on the task template itself. To highlight all the sides of the solution presented, the diagram section is decomposed into three distinguishable parts as depicted in table 1. The benefit of each one of those components as well as its' adaptability are discussed in the following:

a) Task Hierarchy: In this part, the task template itself to be loaded by the developer and integrated in her model is presented in CTT (Paterno, 1997) notation. Due to the various task types and moreover the high expressiveness of the set of temporal operators offered by CTT, it is convenient as a notation for the task models which are to be used as a basement for the application design process. We suggest the usage of decision nodes in the sense they are used in (Wurdel, 2007). This way the context of use can be explicitly visualized on the edges of the task tree, and during instantiation phase, the conditions are being evaluated and only one path is taken. From our point of view these decision nodes are convenient for integrating the context whenever the adaptation attribute is changing the structure itself of the tree. However, we can still have the case that we aim to adapt the pattern to the object to which it is applied without substantial changes in the hierarchy itself. A good example for that is the search pattern rendered in (Sinnig, 2004) where the pattern can be adapted to the object searched for (e.g.: hotel, car, book…etc.). Therefore, we adopt the concept of domain variables presented by Sinnig in order to consider those kinds of contextual variables.

b) Environmental Dependencies: Here, all of the entities which can be considered as environmental dependencies for the execution of any of the tasks within the template and which are categorized as one of the components of the intervening environment (as depicted in Fig.1.) are presented using a UML class diagram (UML, 2011). Three particularly noteworthy points are to be mentioned here. First of all, this class diagram is picturing the needed entities for the performance of the whole task template and not a specific executable task in the tree. Secondly, the diagram is not an attempt to represent and model all resident elements in the environment, but only the obligatory entities required to perform the tasks. Thirdly, a major benefit of this diagram is to give the developer the required information to build the other related models to this task model. (e.g.: device model, object model, etc.). We suggest adapting this diagram using the cardinalities assigned to the embraced elements. For example in the "Present Slides Pattern" if the user is not deaf, then in the instantiation phase a value of "0" is assigned to the entity text to speech converter. Identically, the number of projectors to be used can be decided in the instantiation phase.

c) Execution Constraints Visualization: Unlike the *"environmental dependencies"* field which aims to assist the developer building the related environ-

Table 1: Present slides pattern.

ID	1
Name	Present Slides
Problem	Use the projector and the presenter device in order to present some slides to the audience.
Situation	A given user in the environment has to present some slides on the canvas. This may be needed in a conference session, lecture or a discussion.
Solution	The actor who is performing this task needs to iterate over all the slides of his presentation and to explain them one by one. As a pre-request, he/she should be located in the presentation area, having the slides to be presented stored on his/her presenter device which is connected to the projector in use. The number of projectors needed depend on the presentation mode (e.g.: the smart room gives the user the opportunity to use only one projector for his presentation, or alternatively several ones in case the slides should be presented on more than one canvas). Only in case the presenter is deaf, he/she can use a text to speech converter to present the slides to the audience.
Diagram	I. Task hierarchy:  II. Environmental dependencies:  III. Execution constraints visualization: 
Adaptation variables	number of projectors, kind of user impairment
Referenced patterns	Deaf Output Accessibility

mental models, this field is focusing on the specification of constraints for every single executable task within the task template loaded by the developer. To realize that, we employ the UML activity diagram notation (UML, 2011). The authors in (Brüning, 2008) tackle the idea of bridging the gap between software engineering domain and task models by providing a valid transformation from task models to activity diagrams. A corresponding transformation rule for every temporal operator in CTT to an activity fragment is proposed. We find this idea promising and we extend it here in order to express all the constraints related to every single task to be executed. Therefore, we define every leaf node (executable task) as a simple action in our activity diagram. Then for a given task, two types of constraints can be assigned. First, we have the temporal constraints which express the order of execution of this action relatively to the other ones and which are represented following the rules in (Brüning, 2008). Secondly, we have the environmental constraints related to this action's execution and which we suggest to manifest using notes enabling the assignment of preconditions and effects in a formal way. OCL (Object constraint language) (OMG, 2011) is employed to express those constraints using formal statements. While formal languages are powerful enough to identify precise constraints, they remain difficult for the people to use. The reason we choose OCL is that it is a formal language ruling out any ambiguities and meanwhile it can be easily understood and written by humans. In order to adapt this component, we take advantage of the decision nodes provided by the activity diagrams. In the pattern's definition every context-dependent variable results in a decision node to be added, and while the pattern is instantiated those decision nodes are removed and only the actions to be executed remain in the model by removing the other invalid paths.

Having this overview of the components of the solution presented by a given task pattern in our library, one can see that each of those components is playing a different role by focusing on a specific aspect. While the "task hierarchy" field provides the task fragment itself to be used within the developer's task model, the "environmental dependencies" part is giving necessary information about the other entities needed and thus it helps building the other models. Finally, the "execution constraints visualization" highlights all the necessary constraints for every individual task's execution and all effects resulting from this execution.

3 PATTERN APPLICATION EXAMPLE

In order to make our ideas and the benefit behind our patterns more clear, we provide here an application example for the "Present slides pattern". Let us consider the following scenario taking place in a given smart meeting room: *"A lecture is to be given today by Professor Georges. He enters the room, sets up his equipment, introduces his talk and then starts presenting the slides to the audience. Those slides are presented on 2 canvases to guarantee a good visualization for all the attendee. Once he is finished with his slides, the audiences are allowed to ask questions that should be answered by him. After answering those questions, he unplugs his laptop and leaves the room"*.

According to (Wurdel, 2008), to build a valid CTML model for the scenario we have to start by identifying the roles taking place. Two main roles can be easily extracted from the previous scenario. We have the lecturer role played only by Professor Georges, and we have the listener role played by the audiences. We keep focusing on the lecturer role. The actor playing this role has to enter the room, introduce the topic of his talk and then he has to iterate over the slides of his presentation while explaining them. Afterwards, in case there are questions from the plenum, he answers them before finally leaving the room. Unlike the usual case, in this talk Professor Georges decides to use 2 canvases and consequently 2 projectors. Thus, in the pattern's instantiation phase the number of projectors employed should be settled to 2. However, the scenario did not mention any kind of impairment the presenter is suffering from. Now in order to instantiate the pattern, we have to substantiate every generic part (context-dependent) by the concrete value we have. For this pattern, we have only two adaptation variables which are *"number of projectors"* and "*kind of user impairment*". According to the described scenario, "2" has to be assigned to the first variable and "none" to the second one. For our task hierarchy, the decision node can now be evaluated by removing the "deaf" path. The resulting task structure after "Present Slide" pattern's instance being integrated in the task model is depicted in Fig.2. For the environmental dependencies section,

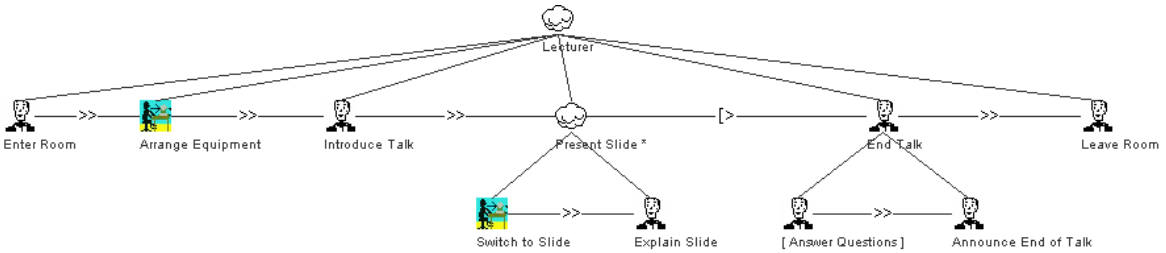


Figure 2: Lecturer's task model after pattern integration.

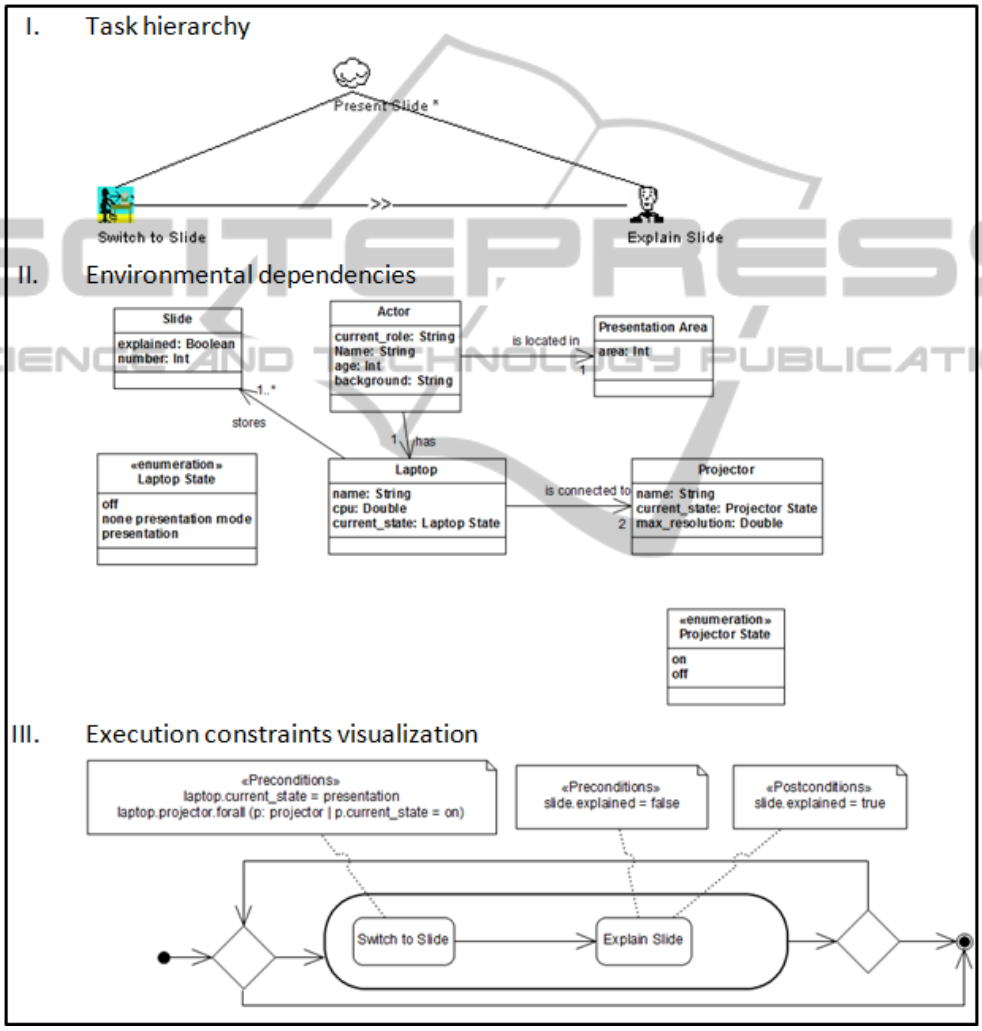


Figure 3: Present slides pattern's instance.

it is clear that there is no need for a converter and so we assign the value *"0"* to this entity (disappears). Additionally, the cardinality of the entity projector is *"2"*. Finally, for the last part which is visualizing the execution dependencies, the decision node branching the default and deaf user cases is also evaluated and the deaf user's path is removed. Moreover, as a precondition for the task "switch slide" two projectors have to be in the state on. The resulting pattern instance to be employed for our model is depicted in Fig.3.

4 CONCLUSIONS

In this paper we presented an attempt to adapt the

concept of task patterns to the field of smart environments. We started by answering the question: Why do we have to model? Afterwards, we highlighted the role that patterns can play in order to make this modeling process faster, of better performance and less-error prone. Afterwards, we made a distinction between the forcing and the flexible contexts and we extended the already known definition of task patterns by integrating the forcing context which is composed out of the task's related preconditions and effects. We argued that taking this concept of context into account enables to facilitate the building process of the other models and additionally makes it feasible to integrate the solution not only as a simple task fragment, but as a building block where all the mandatory related environmental constraints are expressed. We provided a detailed description of the structure of our twenty patterns filling a task pattern library addressing the domain of smart meeting rooms. Also, the level of flexibility offered by the patterns was discussed and we elaborated the idea of taking impaired users into account in order to give the developer the opportunity to design universally accessible applications while being assisted by those patterns. Moreover, one example of those patterns was presented in full details and finally we discussed an application example where a simple scenario was presented and we illustrated the instantiation of this pattern and its' usage to assist the developer while building the task models to be assigned to the included roles. Now, we are developing a pattern tool that we believe can allow the usage of our patterns in an easier and seamless way. As for our approach we investigated CTML as a suitable language for the domain of smart environments, we are extending the CTML editor which provides an Eclipse-based IDE to build task models by our task pattern application tool. We believe that this tool can offer real assistance to the developer while developing applications in the context of smart environments.

REFERENCES

Alexander, C., S. I., Silverstien, M., 1977. A Pattern Language. In: *Towns, Buildings, Construction*, Oxford University Press.

Blumendorf, M., Lehmann, G., Albayrak, S., 2010. Bridging Models and Systems at Runtime To Build Adaptive User Interfaces. *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems.* ISBN: 978-1-4503-0083-4: 9-18.

Borchers, J., 2001. A pattern approach to interaction design, *DIS '00 Proceedings of the 3rd conference on Designing interactive systems: processes, practices, methods, and techniques.* ISBN: 1-58113-219-0: 369-378.

Breedvelt-Schouten, I. M., Paternò, F., and Severijns, C. 1997. Reusable structures in task models. In *Proceedings of DSV-IS: 225-239.*

Brüning, J., Dittmar, A., Forbrig, P., and Reichart, D., 2008. Getting SW Engineers on Board: Task Modeling with Activity Diagrams. In *Engineering Interactive Systems*. *Lecture Notes in Computer Science,* Vol. 4940. Springer-Verlag, Berlin, Heidelberg: 175-192.

Cook, D., Das, S., 2004. Smart Environments: Technology, Protocols and Applications. ISBN: 978-0-471-54448-7.

Feuerstack, S., Blumendorf, M., Albayrak, S., 2006. Bridging the gab between Model and Design of User Interfaces. In *Proc. GI Jahrestagung* (2), pp.131-137.

Gamma,E., H., R., Johnson, R., Vlissides, J., 1994. Design Patterns: Elements of Reusable Object-Oriented Software. *Reading Mass*., Addison-Wesley.

Luyten, K., 2004. Dynamic User Interfaces Generation for Mobile and Embedded Systems with Model-based User Interface Development. *PhD, Hasselt University.*

OMG,(Object management group) Retrieved October 1, 2011 from *http://www.omg.org/spec/OCL/.*

Paternò, F. 2001. Task Models in Interactive Software Systems. *Handbook of Software Engineering & Knowledge Engineering.* S. K. Chang, World Scientific Publishing Co.

Paterno, F., M, C., Meniconi, C., 1997. ConcurTaskTrees: A diagrammatic Notation for Specifying Task Models, in *INTERACT 97,* IFIP TC13: 362-369.

Sinnig, D., 2004. The complexity of Patterns and Model-based Development. *PhD Concordia University,* Montreal.

UML (Unified Modeling Language). Retrieved October 1, 2011 from http://www.uml.org/.

Wurdel, M., Forbrig, P., Radhakrishnan, T., Sinnig, D., 2007. Patterns for Task-and Dialog-Modeling. *Springer Volume* 4550/2007, 1226-1235, DOI: 10.1007/978-3-540-73105-4_133.

Wurdel, M., Sinnig, D., Forbrig, P., 2008. CTML: Domain and Task Modeling for Collaborative Environments. *J.UCS 14: 3188-3201.*

Zaki, M., Forbrig, P., 2011. User-oriented Accessibility Patterns for Smart Environments. *Springer Volume* 6761/2011, 319-327, DOI: 10.1007/978-3-642-21602-2_35.