2013 IEEE 10th International Conference on Ubiquitous Intelligence & Computing and 2013 IEEE 10th International Conference on Autonomic & Trusted Computing

A Run Time Executable Task Model For Ambient Intelligent Environments

Asma Gharsellaoui,Yacine Bellik
AMI Group, LIMSI-CNRS, Paris-South University
Bât 508, Plateau du Moulon, B.P, 133,
91403 Orsay Cedex, France
asma.gharsellaoui@limsi.fr
yacine.bellik@limsi.fr

Christophe Jacquet
Department of Computer Science,
SUPELEC Systems Sciences,
3 rue Joliot-Curie
91192 Gif-sur-Yvette Cedex, France
Christophe.Jacquet@supelec.fr

***Abstract*—Existing task models are static and used only at design time. The information contained in the model is taken into account only when designing the application and will no longer be changed. In this paper we propose to use the task model at runtime, in order to track user actions, verify that he/she has not made any errors while accomplishing his/her tasks and to give help when asked for. In particular, we present a task model specific to the interactions with an ambient environment and we extend the notion of static task models to allow its dynamic update at run time. Our extension consists in giving runtime task states suitable with information received from the environment. Our second contribution is a monitoring and assistance system based on our dynamic task model. We present a simulation of our application based on a software prototype. This application shows how the interactions with the task model at runtime allow us to produce a dynamic and context aware system dedicated to the help of the user to do his/her daily tasks.**

Keywords—executable task model; user-centered systems; user tasks monitoring and assistance; Ambient intelligent environments; proactive system;

I. INTRODUCTION

The emergence of new technologies has profoundly affected everyday life. Ambient environments strive to support users through their embedded intelligent objects [1] and to satisfy their needs based on their preferences and habits. To deal with these sophisticated mediums and to realize correctly their daily tasks, users are in need of supervision and assistance. Supervising the user tasks is to see what they are doing in order to be able to help them making decisions when needed. Task assistance consists in advising users about which action to undertake and providing direct support to them.

In this context our work proposes a novel monitoring and assistance system based on an adapted task modeling approach. Section II summarizes the task modeling requirements in an AmI environment and reviews the existing assistance solutions based on task models. Our main contribution lies in Sections III and VI that respectively introduce our proposal for a task model adapted to assistance in ambient setting, and contain the specification of the monitoring algorithm. Section V presents our simulator through an example scenario. Section VII gives conclusions about our monitoring and assistance system and directions for future work.

II. AMBIENT INTELLIGENCE AND TASK MODELING

A. Ambient Intelligence

An Ambient Intelligent (AmI) environment is defined as an intelligent, digital environment that is sensitive and responsive to the presence of people. An ambient system can act through its embedded actuators or acquire information from its environment sensors or also interact with the user [3]. Sensors may provide clues about the behavior of people (position, orientation, displacement...) that could then be exploited to ensure the proper conduct of tasks performed by the user and to offer assistance when needed.

B. Requirement of task modeling in Ambient Intelligent Environments

The purpose of task modeling is to build a model which precisely describes the relationships between the various tasks [4]. A task model describes the intended activities to be performed in order to reach user goals [5] and the different ways to accomplish them [6].

In a previous work [7], we have conducted a study to identify the requirements of task modeling in Ambient Intelligent environments. The first constraint is that the task model should include a way of labeling tasks with spatial constraints and devices manipulated to accomplish the tasks. Second we have addressed the level of granularity of the task model and we have proposed to stop the decomposition at the level where system services are invoked except in the case of user tasks which do not rely on software services, but on human actions only.

The main characteristic of a task model adapted to ambient environment is that it should be dynamic model. Indeed we need a model updated at run-time in order to respond to the continuous changes of the context.

C. Models and use at runtime

A large number of task models have been developed, especially in the context of GUIs: HTA [8,9,10], GTA [8,11],CTT [12], UAN [13], TKS [14], DIANE+ [15,14,8], TOOD [8,16]. We found that none of the cited task models

978-1-4799-2481-3/13 $31.00 © 2013 IEEE
DOI 10.1109/UIC-ATC.2013.55

IEEE computer society

could be used for modelling tasks in ambient environment. For instance, none of them has a specific notation for device- or place-related tasks since they deal only with tasks invoking only one system. The richest task model for our needs seems to be the ConcurTaskTree or CTT model.

In highly dynamic environments, such as those involved in ubiquitous computing cases, we need to update the task model at runtime to adapt the application to specific circumstances. A number of works have been carried out regarding the real-time adaptation of applications based on task models. In the domain of user interfaces, in [17] authors present an executable task model to create applications adapted to the actions of the user. They extend the CTT notation to allow the dynamic execution of a task model at run-time by keeping the active task state for leaf tasks. Additionally, the authors propose input and output ports to extend the temporal operators to facilitate information exchange. Their application interprets and uses the information conveyed by the task model. They used this executable task model to generate a user interface composed of pre-programmed UI blocks.

One major application of models at runtime is the adaptation of systems at runtime. In [18] the authors propose a method of task and memory assistance for elderly persons using a mobile communication device including storing a profile of a user and determining a task to be performed by the user based on the user profile. This application aims to help persons suffering from cognitive impairments to perform tasks composed of sequential steps involved in a daily routine. Another approach for runtime adaptation based on models is discussed in [19]. In this work, runtime models are used for self-adaptation in the ambient assisted living domain. Interactive components are equipped with an adaptation and configuration model. The adaptation manager monitors context changes in order to decide which system change to choose among a set of predefined possible system changes. This decision is then performed by a configurator component. Another work [20, 21] presents a self-adaptation runtime model for Self-Adaptation in the Ambient Assisted Living Domain. They use models at runtime in order to make the running system aware of the context information. They applied the approach to build adaptive multimodal user interfaces for smart home environments.

III. RUN TIME EXECUTABLE TASK MODEL

Here we will describe our proposed task model which is inspired of the CTT model in order to adapt it to the specificities of AmI environments. The model state is dynamically changing to respond to the continuous changes in the environment. Our monitoring system will be based on the use of this new task model in runtime.

A. Task Tree Structure

Our task model has a binary tree-like form: each task is decomposed into a maximum of two tasks. A hierarchical structure enables a clear identification of task steps necessary to reach a certain goal. We distinguish between two types of tasks: Concrete Tasks and Abstract Tasks. The Concrete Tasks are the leaves of the tree; they represent elementary tasks that can be detected while being performed by the user, the system or both in the case of Interactive Tasks. These tasks cannot be further decomposed.

The abstract tasks reflect the behavior of a temporal operator connecting two subtasks as illustrated in figure 1:

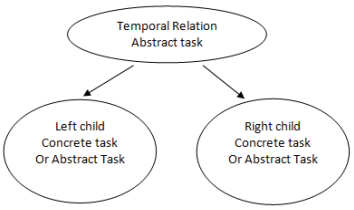

Fig. 1. Example of Tasks Decomposition

B. Annotation of elementary tasks annotation

An elementary task has a number of proprieties which can either be static (once established the information contained will no longer change), or dynamic (their content will change when receiving new information from the environment). We will use “D” to refer to dynamic proprieties and “S” for static ones.

- Identifier or task name “S”.
- Importance “S” {high, medium, low}: this characteristic will be helpful when concurrent tasks need to be prioritized.
- State “D” {Inactive, Possible, Active, Suspended, Done, Stopped}: calculated with respect to the temporal relationships connecting the different tasks (see below).
- Interruptible “S”: indicates whether the execution of the task can be interrupted by another task.
- Service “S”: represents a service invoked by the task. For purely user tasks this field will be empty as there will be no call for services.
- Pre-condition “S”: a condition that must be verified before the task can occur. The prerequisites of the task will include all the necessary conditions for the realization of the task such as being in a specific location or manipulating a specific object.
- Post-condition ”S”: a set of conditions which are known to be verified after the execution of the task, thereby describing its effect on the environment.
  The state of Pre- or Post-conditions will be changed at the reception of a new entry from the environment.
- Obligatory ”S”: determines whether the non-realization can have serious consequences on the environment or to achieve a given purpose.

C. Temporal operators for abstract tasks

The abstract task plays the role of a scheduler between its two children tasks. We used a rich set of temporal operators (the same used in CTT):

- Choice []: the left or right action can take place (only one of them can be performed).

- Order Independence |=|: both of the actions must be performed but in any order, we can start with the execution of the left or the right task.
- Interleaving|||: the two tasks are performed in parallel.
- Synchronization |[]|: the two tasks they start and end their execution at the same time.
- Disabling [>: The left task starts and can be interrupted by the right task but in this case it cannot be resumed any more.
- Suspend-Resume |>: The left task starts and can be interrupted by the right task, once the right task is executed the left task can be resumed.
- Sequential Enabling >>: Once left task is entirely executed we can start executing the right task.
- Sequential Enabling Info []>>: same principle as previous but once the right task ends it passes information needed by the left task.

D. Task states

Since we propose a run-time task model, the task state is updated continuously with the changes in the surrounding environment. The possible states of a task are:

- INACTIVE: The initial state; the task has not yet been performed.
- POSSIBLE: it means that the task can be performed.
- ACTIVE: the task is being performed.
- SUSPENDED: this task was being performed when another task suspended it and its realization will be resumed once the new task is totally executed.
- STOPPED: the execution of the task was interrupted by another one and cannot be resumed.
- DONE: the task was completed entirely with success.

Task states and the transitions between them are the main source of information when trying to capture the current state of a task model. Figure 2 shows the complete set of states including the transitions.

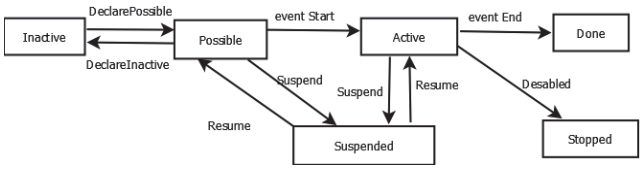

Fig. 2. Possible Task States Represented as a state machine

By default a task is in the INACTIVE state. The states of the different tasks are calculated each time the system receives an event from the environment. A POSSIBLE task changes to the state ACTIVE when the system notices the beginning of the realization of the task. Once we receive an end event reflecting the completion of the task, the task model changes the state of the task to DONE and sets its post-condition to true.

IV. Monitoring and Assistance Algorithm

A. Global architecture of the solution

We propose a Monitoring and Assistance system that is based on the use of the above task model at run time as a reference to what has to be done. This information is compared to what is really actually done in order to give help to the user if he/she omits to do a task. This is presented in figure 3.

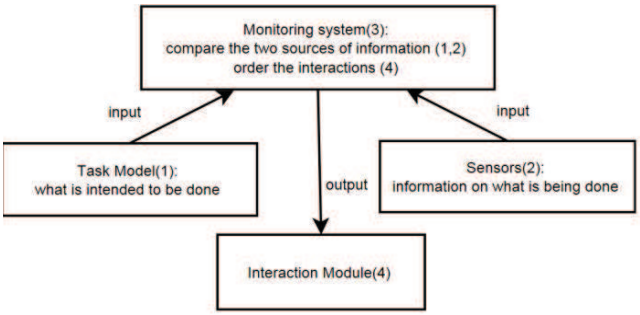

Fig. 3. Global Architecture of the System

The system (3) continuously receives information from the sensors (2) giving information on what is actually occurring in the surrounding environment. This information is compared to the information contained in the task model (1) that describes what the user has to do. If the system notices that the user is doing a wrong task, it orders the interaction module (4) to inform the user that a certain task was not done correctly or that he/she has not done a task.

B. Algorithm description / System Behavior

The system receives in real time information describing what is happening in the environment. We distinguish between two types of entries that can be received:

1) Any information about the processing of a concrete task: if it has started or finished.

2) Any noticed change in the state on the different sensors that could be useful to calculate the state of the pre-conditions.

Each time the system receives information it updates the current state in the run-time executable task model to be able to follow the progress of tasks realization. Once the system receives sensor information it updates relevant pre-condition in the task model.

When receiving an entry of type (1), the system calculates the new state of the task and notifies the parent tasks about this new state. These parent tasks recalculate their own state and the state of their own children. This information is propagated (with respect to the semantics of the different temporal operators between them) until it reaches the root task.

The system must ensure that all the pre-conditions of a task are satisfied before it can start. Pre-conditions can be for example: “At least one device with the service invoked by the task is available” or “the user must be in the realization area of the task” (this area is defined for each specific service using a given device).

The monitoring system refers to the states of the sensors to evaluate pre-conditions. Once a task becomes DONE its post conditions are set to true and the system propagates this change to the pre-conditions of all other tasks since a post condition of one task can be a pre-condition of another.

When the system sets a task as POSSIBLE it triggers the DTE or Deadline Time for Execution count down. We define the DTE as the maximum time duration between the instant the task gets the state Possible until it starts being active. This information is used for assisting the user. Some tasks needs to be completed after a certain period of time and no later as their non-realization could have dangerous effects on the environment. Let's take an example of a cooking task that must not exceed ten minutes. The user must switch off the cooker ten minutes after turning it on, this time must not be exceeded otherwise the meal will be burned. The system must instruct to the user to do the task before this period of time expires. This alert must be early enough so that the user can notice it and act before the deadline.

Consider a task Ti having a set of pre-conditions called Prec. T_{exec} is the time needed to accomplish a task or a pre-condition. The deadline time for execution of the task Ti is equal to the time needed to accomplish the longest pre-condition of Ti added to the time needed to notify this change to the system called $T_{notifChange}$. DTE could be formulated as follows: $DTE(Ti)= \max_{p \in Prec} [T_{exec}(p)+ T_{notifChange}(p)]$
In some cases the pre-condition "p" could be a post condition of another task Tj so we need to wait for the execution of Tj to get the pre-condition verified. This leads us to this second definition: $T_{exec}(p)= T_{exec}(Tj)+ T_{notifChange}(Tj)+DTE(Tj)$

Knowing the approximate time of task realization we will be able to establish the task planning in advance. The monitoring system will be waiting for the realization of the task pre-condition and at the same time will look for a substitution system action that could verify the pre-condition.
If there is a system action able to verify this pre-condition, the system performs it and waits for the start of the task. If the pre-condition cannot be verified by a system action, the user has to be alerted that a certain condition has to be verified. If we continue with the cooking example, the user task is to switch off the cooker; the pre-condition of this task is to be in the kitchen. If the system notices that the user is not in the kitchen in time, it cannot change the user place but only alert him to move to the kitchen to be able to realize the task.
Figure 4 presents an overview of the monitoring algorithm.

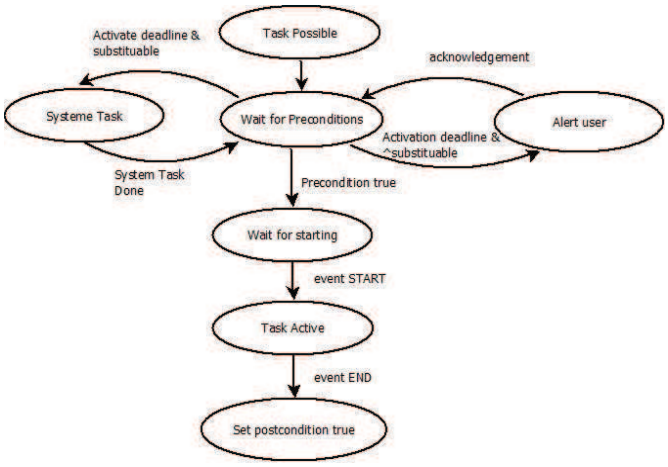

Fig. 4. Assistance and monitoring Algorithm

V. THE SIMULATOR

A. *Ambient scenario "Daily tasks"*

Bob starts his daily tasks. He can either start with preparing foods or cleaning the house. Both of these two tasks can be interrupted by a phone call or someone knocking at the door.

To prepare lunch he must start by cooking pasta: first of all he takes the pan out at the cupboard; he puts water in it and set it on the cooker. Once the water is boiling, he can put the pasta in the pan. He tastes the pasta and finds them ready so he takes them out of the pan and drains the water.
Now that the pasta is ready he has to prepare the sauce, first he starts peeling an onion. At this moment the house phone starts ringing; he goes to answer the call and suspends his current tasks. Once he finishes the call, he comes back to the kitchen, cuts the onion into small pieces, takes the stove out of the cupboard and puts the onion in it. He adds vegetable oil and puts the mixture in the cooker and 4 minutes after he pours sauce. Ten minutes after the system notices that he didn't come back to put off the cooker. The system instructs him to take off the sauce and to put off the cooker. He comes running and did what he forgot to do.

Once the lunch is ready he turns to the cleaning of the house. He starts with ordering beds and dusting the furniture. At this moment his son comes back from school and knocks at the door, so he interrupts his activity, opens the door for him and comes back to clean the floor with the vacuum cleaner before starting to wash the dishes and washing clothes.

B. *Simulator*

To validate our algorithm we have developed a simulator that takes in input a task tree built with the Concur Task Tree Environment (or CTTE) [22]. This tool will be used to build task models (see figure 5).

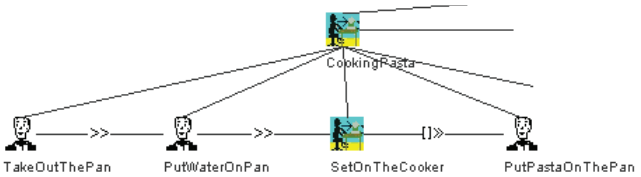

Fig. 5. Example of a CTTE task tree

We have developed a parser that performs a depth-first traversal of the task tree. The generated task model is a binary task tree built with respect to the different temporal relations and their priorities. Figure 6 is an example of how a CTT model is transformed into one of our binary trees.

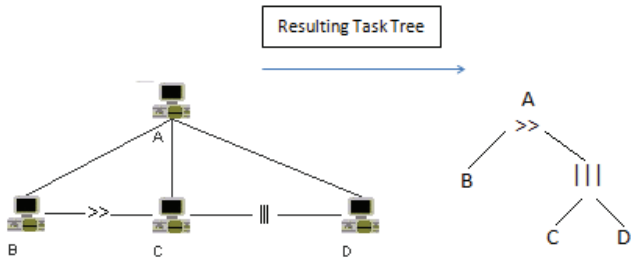

Fig. 6. From CTT model to our presentation

For each level of the input task tree, we will store the temporal operators and restructure the task tree referring to their priorities. For each concrete task we will store a copy of the information extracted from input task tree. We also store for each task the services addressed, the possibility of interrupting it, the deadline of execution and the set of its pre- and post-conditions. It is not possible to add such information in the CTT model except the pre- and post- conditions that are already included but they use a specific syntax that doesn't allow us to express all possible conditions.

We use CTT's plain text "description" field to add specific information.

Pre-conditions are written in propositional logic. For example: ((precd1 | precd5) & !precd6)

- | or + represent the operator OR
- & or * represent the operator AND
- ! represents NOT.

Figure 7 shows a sample of our model binary tree like form.

```
A- DailyTasks -> SuspendResume (POSSIBLE)
   A- null -> SuspendResume (POSSIBLE)
      A- null -> OrderIndependence (POSSIBLE)
         A- PreparingFoods -> SequentialEnabling (POSSIBLE)
            A- null -> SequentialEnabling (POSSIBLE)
               A- PerparePast -> SequentialEnablingInfo (POSSIBLE)
                  A- null -> SequentialEnablingInfo (POSSIBLE)
                     A- null -> SequentialEnabling (POSSIBLE)
                        A- null -> SequentialEnabling (POSSIBLE)
                           C- TakeOutThePan ( state: POSSIBLE **** precondition: (`CupboardOpen` & `PanOn
                           C- PutWaterOnThePan ( state: INACTIVE **** precondition: ((`PanOnHand` & `Open
                        C- SetOnTheCooker ( state: INACTIVE **** precondition: (`CookerOn` & `FreePlaceOn
                     C- PutPastOnThePan ( state: INACTIVE **** precondition: (`WaterBoiling` & `PanOnTheC
               C- DrainThePasta ( state: INACTIVE **** precondition: (`WaterLevel<Min` | `PastaReady`)
            A- preparer sauce -> SequentialEnabling (INACTIVE)
```

Fig. 7. Our Binary Task Tree –initialization phase-

"A" means it's an abstract task and "C" means it's a concrete task or a tree leaf that is an observable task. Each task has its state between the first brackets, inactive is the initial state.

The environment changes can be communicated to the monitoring system using this window:

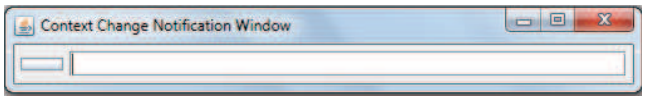

Fig. 8. Context Change Nofitication Window

Two types of entries can be received:

- TaskID START/END : The taskID refers to the name of the concrete task, the event Start means that the execution of the task has started and the event End means that the task was performed entirely with success.
- Info Pre-condition state (true/false): This second type of entry gives information provided by the different sensors. The system is informed each time that we notify any change in the environment. Any information concerning the pre-conditions received by the system is communicated to all the pre-conditions containing this condition.

If the pre-condition of a task is verified (as shown in figure 9), it becomes feasible and it can start operating.

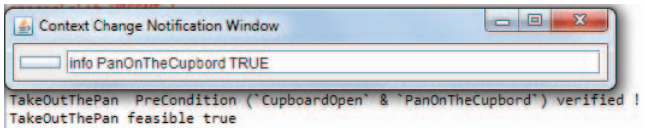

Fig. 9. Example of a task change to feasible

Now that the pre-conditions are verified, we notice that the task "TakeOutThePan" is started, this information is provided to the system which has to recalculate at run time all the task states with respect to the temporal operators connecting them.

At this stage, we can receive an event end on the active task or information on any pre-condition state. Let's consider that we had the three pre-condition of the next task true, the result will be shown in Figure 10.

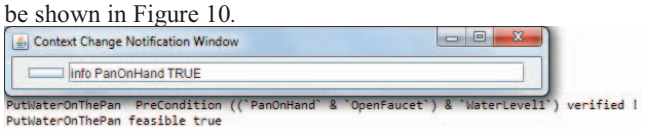

Fig. 10. Example of entering a condition change state

And now let's see what happens when we receive an event end on the active task: the task state changes to done, the next task takes the state possible and the post-conditions of this task are set to true (see figure 11).

Once a task DTE remaining is becoming very short, the system will show a red message to the user telling him/her that this task is urgent and he/she must proceed to its execution.

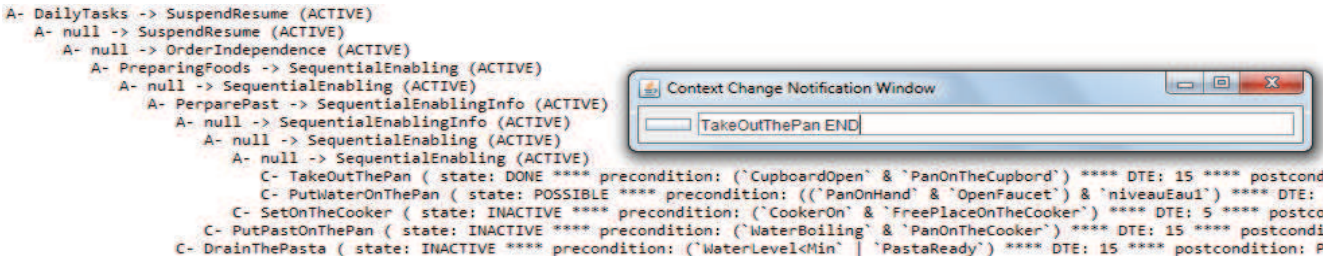

Fig. 11. Task States updated at the reception of an event end and post condition set true.

VI. CONCLUSION

In this paper we have proposed a task model adapted to ambient environment that extends the existing CTT notation and semantics and its state can be updated in real-time. Our extensions include the definition of different levels of task states for concrete tasks (leaf tasks) and abstract tasks (parent tasks). Also we have defined the information exchange between parent and children tasks with respect to the semantics of the temporal operators while moving from one task state to another.

We have further shown how a monitoring and assistance system can use such an executable task model to dynamically follow the execution of the tasks, according to the status of the task model and knowing what is really occurring in the environment from the information given by the sensors. Our system automatically inherits the correct temporal behavior according to our model.

The example simulator used in this work is simple. We are currently investigating a real-scale validation of our monitoring system based on our dynamic task model on our intelligent room at our lab.

Our research on using task models at runtime is still in progress. The executable model is meant to be a prototype that further research can be based upon. We plan to deploy our monitoring system in a real application case in order to see whether it is helpful for users. Future works will focus on the determination of the best strategies for interaction with a user since a silent system is not useful and a system with a lot of notification is also embarrassing to the user and could disturb him/her while doing his/her tasks.

REFERENCES

[1] G. Riva, M. Lunghi, F. Vatalaro, and F. Davide, “Presence 2010: The Emergence of Ambient Intelligence”, in: G. Riva, W.A IJsselsteijn (Eds.), Being There: Concepts, effects and measurement of user presence in synthetic environments, IOS Press, Amsterdam, 2003, pp. 60-81.

[2] A. Gaggioli, “Optimal experience in ambient intelligence”,in Ambient Intelligence, IOS Press, 2005, http://www.ambientintelligence.org.

[3] G. González, , A. Cecilio, B. Lopez, , and de la Rosa “Smart user models for ambient recommender systems“. In Ambient Intelligence and (Everyday) Life., University of Basque Country, San Sebastian, Spain, pp. 113-122. , 2005.

[4] F. Paterno, “Task models in interactive software systems”, Handbook of Software Engineering and Knowledge Engineering, Vol 1, 2002, Publisher: World Scientific, pp 1-19, ISBN: 981024973X, DOI: 10.1002/9780470757079.ch4.

[5] E. Schulungbaum, “Model-based user interface software tools current state of declarative models”,1996.

[6] T.C. Ormerod, A. Shepherd, “Using Task Analysis for Information Requirements Specification: The Sub-Goal Template (SGT) Method”, The Handbook of Task Analysis for Human-Computer Interaction. Mahwah, NJ.: Lawrence Erlbaum Associates, 2004, pp. 347-365.

[7] A. Gharsellaoui, Y. Bellik, C.Jacquet. “Requirements of Task Modeling in Ambient Intelligent Environments”. AMBIENT 2012, the Second International Conference on Ambient Computing, Applications, Services and Technologies, September 2012. pp. 71-78. IARIA. ISBN: 978-1-61208-235-6.

[8] J. Guerrero-Garca, J. Vanderdonckt, J.M. Gonzalez-Calleros. “Towards a multi-user interaction meta-model”, Working paper 08/28, 2008.

[9] D. Diaper, N. A.Stanton, “The handbook of task analysis for human computer interaction”,2004, pp.67–116.

[10] S.Mills. “Contextualising design: Aspects of using usability context analysis and hierarchical task analysis for software design”, Journal of Behaviour & Information Technology archive, Vol 26, Issue 6, November 2007, pp 499-506.

[11] Gerrit C. van der Veer , Bert F. Lenting , Bas A.J. Bergevoet. “GTA: Groupware task analysis - modeling complexity”, Acta Psychologica, 1996, vol 91, pp.297-322

[12] P. Rigole, T.Clerckx, Y. Berbers, K. Coninx. “Task driven automated component deployment for ambient intelligence environment”, Pervasive and Mobile Computing, vol 3, Issue 3,2007, pp.276-299, ISSN: 15741192, DOI: 10.1016/j.pmcj.2007.01.001.

[13] J. Coutaz, F. Paterno, G. Faconti, L. Nigay. “Comparison of approaches for specifying multimodal interactive systems”,1994.

[14] W3C group,“Task Meta Models”, February 2010, Online: http://www.w3.org/2005/Incubator/model-based-ui/wiki/Task_ Meta_Models.

[15] M. Barthet, F., & Tarby, J.-C, “The Diane+ method”. In J. Vanderdonckt, (Ed.), Computer-aided design of user interfaces,1996, pp. 95–120. Namur, Belgium: Presses Universitaires de Namur.

[16] F. Moussa, M. Riahi, C. Kolski, M. Moalla. Interpreted petri nets used for human-machine dialogue specification,1st ed., vol. 9. IOS Press: Integrated Computer-Aided Engineering, 2002, pp.87–98.

[17] T.Klug, J. Kangasharju, “Executable Task Models”, TAMODIA 2005, 26-27 Septembre.

[18] A. Helal, C.Giraldo, W.Mann, “Daily task and memory assistance using a mobile device”, US Patent 20050057357 A1, Mar 17, 2005.

[19] D.Schneider, M.Becker, “Runtime Models for Self-Adaptation in the Ambient Assisted Living Domain”, 3rd International Workshop on Models at Runtime at MoDELS’08,2008.

[20] G.Lehmann, M.Blumendorf, F.Trollmann1, S.Albayrak,“ Models in Software Engineering”, 2011, pp 209-223.

[21] M.Blumendorf, G.Lehmann, S.Albayrak, “Bridging Models and Systems at Runtime to Build Adaptive User Interfaces”, EICS '10: Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems, 2010, Berlin, Germany.

[22] F. Paterno, “Task models in interactive software systems”, Handbook of Software Engineering and Knowledge Engineering, Vol 1, 2002, Publisher: World Scientific, pp. 1-19, ISBN: 981024973X, DOI: 10.1002/9780470757079.ch4