Towards a Runtime Evolutionary Model of User Interface Adaptation in a Ubiquitous Environment

Imen Ismail, Faouzi Moussa
CRISTAL Laboratory
National School of Computer Sciences
Manouba University 2011 Tunis, Tunisia
imen_ismail@yahoo.fr faouzimoussa@gmail.com

***Abstract*—Ubiquitous environments are often considered highly dynamic environments and the contextual information can change at runtime. User interface should provide the right information for the right person at the right time. Certainly, such objective can be achieved only when we deduce the realtime user's requirements in terms of information and present this information to the user according to his current context of use. The specific goal of our research is to improve the adaptation process while improving models at runtime. A fixed model cannot handle the high dynamic in such an environment. The model can progress and change its structure to better deduce the user's requirements. The work reported in this paper introduces a pertinent solution for representing the dynamic construction of a Petri-nets based model. The solution applies the ontology of service (OWL-S), given that the contextual information is defined by the ontology written in OWL.**

Keywords-Ubiquitous computing; User Interface Adaptation, OWL; OWL-S; Ontology; Modeling; Petri-nets .

I. INTRODUCTION

User interface adaptation to the context of use is an area of research that is rapidly expanding. The potential progress of the fourth generation networks and technologies such as wireless networks (LAN WiFi, UMTS, Bluetooth, GPRS and RFID) as well as sophisticated, portable, computing, devices such as PDAs, iPhone, iPod, Pocket PC, Wearable computers, etc. are the challenge of researches in user interface adaptation [1]. This area is becoming increasingly complex [2][3]. In a pervasive environment, the user is in front of a wide range of information and heterogeneous content. The aim of making interface more "attentive" and aware of the user's needs have advanced applications. Models used for the user-device interaction should be built with context-awareness capabilities, so that they can properly adapt to the changing context of a moving user. In fact, user interface must adapt the information it provides by implicitly deriving the user's requirement from his context of use, whereas, the context tends to vary at runtime in a highly dynamic environment. In this paper, we are concerned with an approach for modeling the basic components of ubiquitous computing system, i.e.: the user, the user's behavior and their activities. The specific goal of our research is to enhance the adaptation process while improving models at runtime. A fixed model cannot handle the very high dynamic aspect of such an environment. The model should progress and change its structure to better deduce the user requirement. Selecting the appropriate model is not that easy. To address this problem, adaptation strategies will be based on evolutionary models. We will describe these models and how they evolve over time. First, the paper introduces a method to express the user's behavior and therefore find functionalities of user interface. The user's activities were first modelled with Petri-nets technology. User's interaction with the interface was modeled too and finally, we can deduce a user's requirement at every functional step of the ubiquitous computing system. So, the question to be answered here is how can we improve such models in order to perform better in a very dynamic environment. This paper addresses these issues and proposes a method to build realtime models. Note that ontology was used to describe the ubiquitous environment in general and the contextual information in particular, the ontology of service has been passed in order to represent the model's building. In the following sections we reviewed the technique for the user interface modelling. Then, we outlined the solution for the Petri-nets modelling using OWL-S properties. We presented a very simple example developed using our approach. Finally, the Section 4 summarizes this study.

II. MODEL-BASED USER INTERFACE ADAPTATION

Nowadays, one of the main key features of Human Interaction Systems is the adaptation concept [1][4]. Adaptation is defined as a customization process which takes into account all information and parameters. This is often called the context of use, which could be useful and relevant to improve interface usability model [3][5][6]. On the other hand, model based user interface has received much interest. Essentially it consists of an alternative paradigm for constructing interfaces [7][8]. Developers must write a specification in a specialized high level language, such as state transition diagrams [9], grammars [10], or event-based representations [11]. The specification is automatically translated into an executable program; the specification can also be interpreted to automatically generate the user interface. The proposed approach focuses on realtime modeling of the user interface in a ubiquitous environment.

A. Realtime Modeling of the User Interface in a Ubiquitous Environment

In this section, we introduce briefly the model-based approach proposed to support realtime adaptation of the user

 ISBN: 978-1-61208-171-7

interface in ubiquitous environment [12]. As first step, we are concerned with the user's activities because they provide relevant information on what a user is doing. Consequently, the system can deduce the suitable user's requirements to fulfill the current activity. To achieve this objective, this latter concept must be analyzed and modeled. As formal modeling, the Petri-nets technology was chosen, in particular the Interpreted Petri-nets (IPN) [13][14]. This extension of Petri-nets introduced the notion of events and conditions as well as the notion of actions. We associated a passing condition (*Cj*), a triggering event (*Evj*) and a possible action (*Aj*) to each transition (*Tj*). A user's activity is composed of a set of elementary actions. The elementary actions were modeled by elementary structures of IPN (Figure. 1.a).

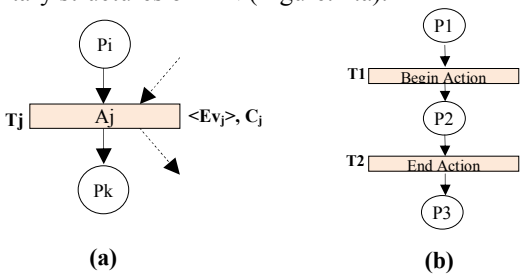

Figure 1. IPN modeling (a) Elementary activity modeling (b)

Having modeled an elementary activity with an IPN, the user's behavior which consists of a set of activities can be modeled (Figure. 1.b). The places represent the user's behavior according to the system's evolution and changes. The validation of the condition i (transition T1) and the presence of the event "*End action*" (transition T2) indicate that the action has been executed and has come to end. The place P2 expresses a waiting state, while the places P1 and P3 model the state of the user before and after the execution of the action. Specifically, the place P3 indicates the end of the action. A behavior can be described as a set of typical compositions of actions, such as sequential, parallel, choice, iteration, etc. Thus, a behavior model can be built by composing all the elementary actions as depicted in the following figures (Figures. 2, 3 and 4). We consider a transition "*Begin Action*" in the user's behavior model. We associated the adequate parameter(s) to these transitions. These parameters refer to the user's requirements at this point of functioning [12]. For example, at the state P2 (Figure.2), the user has the relevant information to well perform his current action, i.e. the Usual Glucose Rate (UGR). Once these parameters have been identified, we can deduce the necessary components and widgets of the user's interface [12].

B. Case Study and Problem Identification

We proposed an application for monitoring diabetic patients in a U-Hospital (i.e. Ubiquitous Hospital). We describe a simple simulation of a medical intervention in a diabetes service. In this example we show how the medical treatment model should change to provide proper information to paramedics, according to the patient's glucose rate. Monitoring the realtime evolution of patients' state and specifically their glucose rate (GR) is the main objective of this ubiquitous system. Real-time monitoring of the patient status can be achieved by using wired or wireless sensors and actuators that periodically control the glucose rate. In the light of those values, the system must verify and record the evolution state of each patient. Therefore, the medical intervention depends on patients' recorded state. Generally, this operation is based on a so-called "action plan". An action plan describes the necessary steps to treat a specific diabetic patient's case.

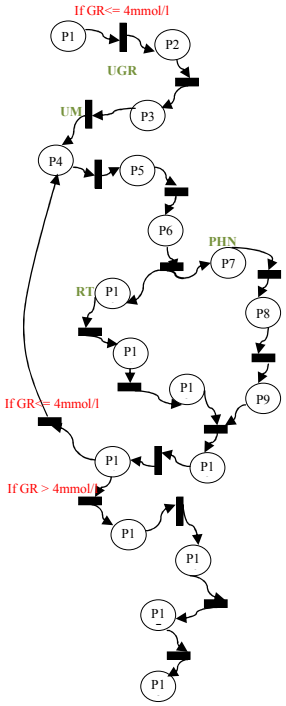

Figure 2. User's behavior model within the user requirements (facing conscious patient's case)

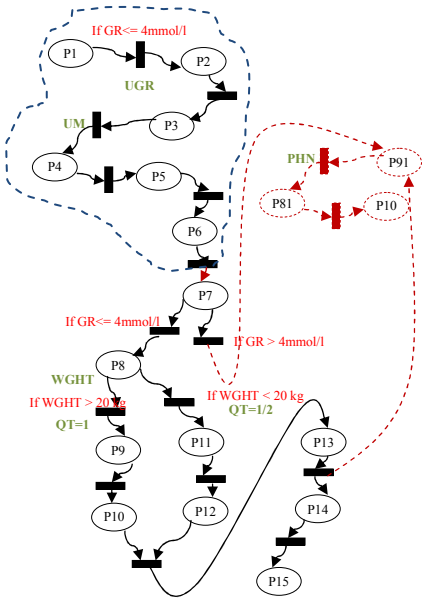

Figure 3. Suitable model to be generated

In this article, hypoglycemia is particularly studied. Both two cases are considered here: the patient is either conscious or unconscious and unable to swallow. Let us assume that, at some point in time, the management system detects an abnormal glucose rate of a given patient (e.g., GR<4mmol/l). It presumes that this value indicates a hypoglycemia with a conscious state. Consequently, the system initiates its functioning based on the associated model (i.e. Figure. 2). Now, suppose that at time *"t"* the patient lost consciousness. At this moment, the current model is not suitable for a better deducing of the nurse's requirements in terms of information. Hence, the system must generate the appropriate model (Figure. 3) or readjust the current one (Figure. 4). The following section describes the runtime architecture for generating evolutionary models and where both the functional components and the semantic representation are described.

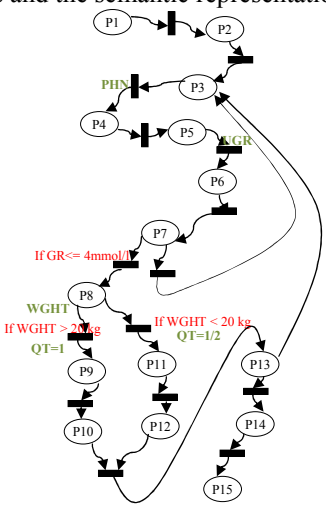

Figure 4. User's behavior model within the user requirements (facing unconscious patient's case)

III. TOWORDS DYNAMIC AND EVOLUTIONARY MODELING FOR USER INTERFACE ADAPTATION

Our approach is specifically based on the variation of the user's activities according to the context. User is involved in a variety of activities over the course of his work. These activities can be routine activities, unexpected activities; they can be also activities that change according to working conditions, etc. In any case, the activity solicits specific information in order to be accomplished. Thus, the models should be built in proportion as runtime evolving of ubiquitous system as well as according to available information. The purpose of this work is to ameliorate the adaptation process while improving these models at runtime. Hence, the model can progress and change its structure in order to better match the user's requirements. In the following section we describe in brief the functional architecture used to model the user's activity at realtime.

A. Functional Description of the Proposed Approach

The intended architecture, given in Figure 5 is mainly based on the "Dynamic Model-developer" module. This is the core architecture considered as the adaptation's engine. It's responsible for assembling and providing the realtime models of the user and then deduces his requirements.

In addition, the objectives of this module are: loading the appropriate functional model for the given user according to his current activity, changing models as required, interpreting models at runtime and ameliorating models based on adaptation strategies. These functions are accomplished with a set of other modules. We cite the module of knowledge storing: the "*Database of user's behavior models*". It includes all information about the users i.e. their preferences, their interests, their activities, and any data that characterizes users. Its content will be increasingly enriched while memorizing and learning the user's interactions and behaviors, in particular the achieved activity and the current one. Then, based on this knowledge, the Dynamic Model-developer module loads or synthesizes the suitable models by selecting the appropriate one or combining others. This combination is based on the "*Control structures and rules*" module. This module encompasses rules and strategies for combining and developing models from elementary models. The functionality of this tool is mainly based on a method proposed by Moussa et al. [14][15] within the context of the specification of human-machine dialogue for interactive process control applications.

The "*Meta-modeling structure*" module is an abstraction of a high level representation of the whole system, in particular the users and their interactions. This module includes abstract models based on formalism of Petri-nets modeling. For example, the core architecture can generate a concrete model of the user's activity or change the state of the model related to the activities. This module will be discussed with more details in future papers.

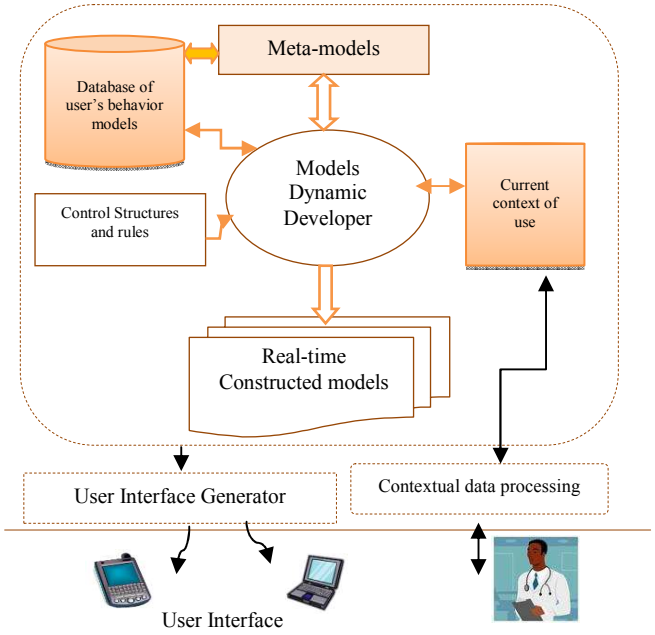

Figure 5. Functional architecture of runtime user interface modeling

The "*Current context of use*" module is supposed to be the data provider. It must perceive the users and their interaction with the environment. It should also notify the core architecture about any change in the entire environment, in particular the users' activities. This information is obtained from the contextual data processing. This component is responsible for processing information, making filters and eventually deducing the context depending on its dynamic execution.

As a conclusion, it should be apparent from our architecture that the system could have, at runtime, a suitable representation of the user in a particular situation. Hence, it can deduce the suitable model or build a realtime model according to the analysis of the user-system's interaction and the context of use. The dynamic models can be created while integrating progressively a range of elementary actions.

B. *Towards a Runtime and Evolutionary model Construction Based on Interpreted Petri-nets Technology and Using OWL-S*

As already mentioned, the problem lies in the realtime construction, or rather a composition, of the user interaction model on the assumption that the sequence of actions cannot be known in advance. The ultimate purpose is to find a pertinent solution for representing a dynamic Petri-nets based model and a dynamic composition of Petri-nets based modeling. Taking into account that in our context specification, the contextual information is defined by the ontology written in OWL (Ontology Web Language); the plausible solution expected to fulfill our requirements seems to be the OWL-S technology [16][17]. In fact, this technology implements the concept of services and provides a representation of services through a process mechanism. If the process is a non-decomposable service, then it is an *atomic process*. Otherwise, it's considered a *composite* service when it includes a set of processes within some control structures. When it deals with a service abstraction, a process is called *simple process*. In this paper, our aim is to adopt this technology to represent a Petri-nets based activity. An atomic process is used to model the elementary action and a composite process models a user activity. The following section gives a brief representation of the OWL-S features and properties. Then, with OWL-S as starting point, we give a description of dynamic and evolutionary model's construction based on interpreted Petri-nets technology followed by an illustration with a simple example.

1) The OWL-S Description Language: OWL-S is an OWL-based web service ontology. It supplies a core set of markup language constructs for describing web service in computer-interpretable form [16][17]. Each service is characterized by three main concepts: Profile, Process and Grounding (see Figure.6).

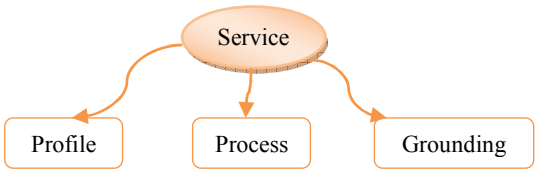

Figure 6. Representation of the service ontology

The profile feature describes the semantic properties and capabilities of a service. It represents a specification of what functionality is provided by the service through a certain number of parameters. The process represents the current composition and gives a detailed description of a service's operation [17]. Finally, the grounding property provides details on how to interoperate with a service via messages [18]. As for functionality description, we quote some properties such as *hasInput* (resp. *hasOutput*) property which ranges over instances of inputs (resp. outputs) as defined in the process ontology. Inputs are information required for the execution of the service, whereas outputs are information that the process returns to the requester. The *hasPrecondition* property specifies one of the preconditions of the service and ranges over a precondition instance according to the schema in the process ontology. The preconditions are determining factors imposed over the inputs and that must hold for the process to be successfully invoked. The *hasResult* property specifies one of the results of the service as defined by the result class in the process ontology [16][17].

Depending on the complexity of the interaction with a service, two classes of services can be identified: Atomic Service and Composite Service [16]. With the former type, a single program, sensor or device is invoked by a request message. Then it performs its task and produces a single response to the requester. Thus, there is no ongoing interaction between the user and the service. Whereas the latter type is composed of more multiple primitive services and may require an extended interaction or conversation between the requester and the set of services that are being utilized.

2) Modeling Services as Processes: A service model exposes how a service works and identifies how to interact with it [16]. Thus, the model views interaction of the service as a process. In other words, a process is a specification of the ways a client may interact with a service. As mentioned previously, the atomic process is a description of an atomic service, i.e. it involves a single interaction to be executed. It's directly called by passing to it the appropriate messages. It takes an input message, does something and returns output messages. The composite processes are decomposable into other processes (atomic processes or composite ones). Their decomposition can be specified by using control constructs: *Sequence*, *Split*, *Choice*, *Any-Order*, *Condition*, *If-Then-Else*, *Iterate*, *Repeat-While*, and *Repeat-Until*. Each control construct is associated with an additional property called components to indicate the nested control constructs from which it is composed, and their ordering.

 ISBN: 978-1-61208-171-7

A process has two sorts of purposes. First, it can generate and return some new information based on information it is given, as well as the world state. Such information production is described by the inputs and outputs of the process. Secondly, it can produce a change in the world. This transition is described by the *preconditions* and *effects* of the process. In fact, *effects* are changes in the state of the world. Moreover, when an *inCondition* property is satisfied, there are properties associated to this event that specify the corresponding output with output property. For additional details, the reader is invited to refer to the OWL-S documentation [16][17].

3) Interpreted Petri-nets Modeling Using OWL-S Properties: This subsection exposes a method proposed in order to match a Petri-nets based model to OWL-S representation. Specifically, the key idea of the intended method is to formulate a Petri-nets based elementary action by using an OWL-S atomic process. And then, to formulate a Petri-nets based activity while composing progressively a set of elementary actions. The idea is based on the hypothesis that the elementary action is a non-decomposable action. The activity is composed of other elementary or other composite actions through the use of compositions rules. These rules dictate the order and conditional execution of the action in the model.

a) Elementary Action Representation: Considering a general representation of the elementary action (Ai) in an activity (Figure.7), the place Pai expresses the input place; the place Pci expresses the output one. Pbi represents the action's place. Once the transition (T1) is firing and the associated condition (Condition i) is verified, the elementary action will arise. Note that the place Pbi models the action's execution. The firing of the transition (T2) allowed moving up from the execution state (Pbi) into the next step: the end execution state (Pci). Thus, to characterize an elementary action the above-cited parameters must be identified. In this situation, we distinguish two types of parameters: those that characterize the beginning of the action (considered as inputs) and those that characterize the end of the action (considered as outputs). Hence, analogously the elementary action can be represented through the atomic service description.

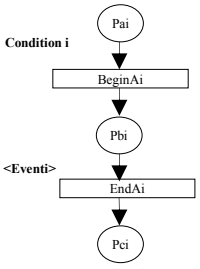

Figure 7. General Elementary Action

The input parameters are necessary information for the successful accomplishment of the action (Figure. 8). They are mainly:

- *Condition i:* a passing condition that must be verified to start the action.
- *BeginAi:* is the triggering of the transition, in consequence starting the action *Ai*.
- *Eventi:* the presence of this event expresses that the action has been executed and has come to end.
- *P*ai: models the state of the user before the execution of the action (*input place*).

The output parameters constitute the information extracted and generated by the action which was performed (Figure. 8):

- *EndAi:* indicates the end of the action *Ai*.
- *P*ci: models the state of the user after the execution of the action (*output place*)
- *Requirements:* a set of contextual parameters that constitute the appropriate set of informational parameters for each transition.

Other relevant information can characterize an action such as:

- *ActionName:* Indicates the name of the action
- *ActionGoal:* Denotes what functionality will be provided by this elementary action.
- *S :* situation (*of the execution of the action*)
- *t :* time (*of the execution of the action*)
- *P*bi*:* The place Pbi indicates a waiting state.

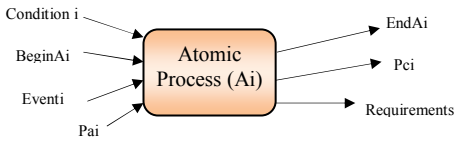

Figure 8. OWL-S based representation of the elementary actions as atomic processes

Table1summarizes a representation of the listed properties while taking full advantage of the OWL-S atomic service description.

TABLE 1. IPN BASED ELEMENTARY ACTION AS OWL-S ATOMIC SERVICE DESCRIPTION

IPN based elementary action	OWL-S atomic service description
Condition i	*hasPrecondition*
BeginAi	*hasInput*
Event i	*hasInput*
Requirements	*hasOutput, hasResult*
EndAi	*hasOutput*
PAi	*hasInput*
PCi	*hasOutput*
ActionName	*ServiceName*
ActionGoal	*hasLocal*
S, t and Pbi	*serviceParameter (Local parameters)*

b) Activity representation: an activity is a set of elementary actions arranged to typical compositions as

sequential, parallel, choice, iteration, etc. [19]. Developing the overall model of the user's activity is based on operational compositions of elementary actions models and on a well-defined composition's rules. Analogously with the elementary action that can be specified with the atomic process, an activity can be represented by the composite process. In fact, we notice that it fits nicely with the composite process and a Petri-nets based activity. This is made possible thanks to many features of the OWL-S description, such as the control constructs that can ensure the composition of elementary actions. The following is an example that illustrates the description of these action's properties through some of OWL-S concepts.

c) Case Study Illustration: As an example, we present the sequential composition of elementary actions. Generally, the sequential composition of N elementary actions is done by merging the output places of the action i, and the input places of the action i+1 (Figure.9). Suppose that a user's activity (*ActivityK*) is composed by the sequencing of two elementary actions *Ai* and *Aj*. Considering the actions A1 and A2 from the action plan [12]:

- *Elementary action A1: GlucoseRateMeasure*
- *Elementary action A2: FirstEmergencyProceeding*

A description of inputs and outputs for each of the atomic processes is required (Figure.10&11). A1 and A2 are instances of the elementary action process.

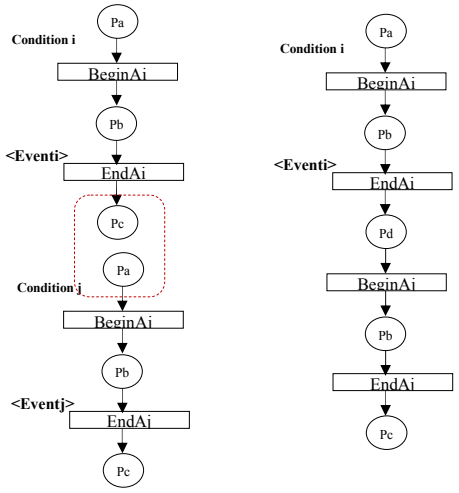

Figure 9. Sequential composition of elementary actions

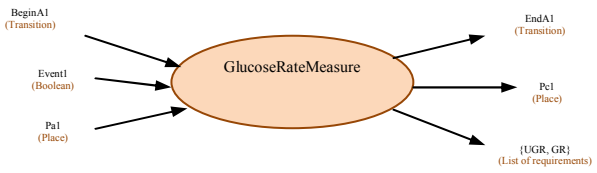

Figure 10. Inputs and Outputs of GlucoseRateMeasure atomic process

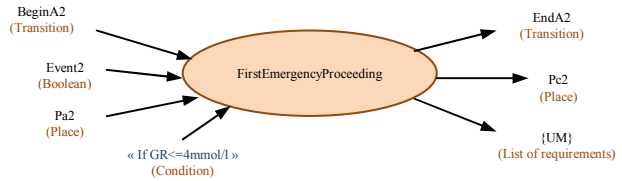

Figure 11. Inputs and Outputs of FirstEmergencyProceeding atomic process

The expressions written in brown represent the type of the inputs and outputs of the elementary actions. We now proceed in describing the construction of the composite process which consists, in general, of the created atomic processes. We consider the whole activity that represents the composite process. We call this activity *unconsciousPatient-InterventionActivity* and the associated process *unconscious-PatientIntervention,* which is a composite processes that consists of A1, A2, ..., A5 actions. We are considering solely the A1 and A2 processes, which are sequential processes. In this case the control construct used is Sequence. The sequence control construct dictates that a list of processes is done successively. Then, a composite process must have a *composedOf* property by which is indicated the control structure of the composite, using a *ControlConstruct*. Then, the data flow specification must be defined. In fact, in many cases when a process is performed as a step in a larger process, there must be a description of where the inputs to the performed process come from and where the outputs go [16][20].

As described previously, the global model of an activity is elaborated using the different elementary actions composed through the control constructs. For this reason, we are going in one hand, to assemble together A1 and A2 by merging the output place of the action A1 (i.e. P_{c1}) and the input place of the action A2 (i.e. P_{a2}) in one place P_{d1}. In the other hand, the possible parameters that must pass from their source to the destination action must be specified. In our example, UG and UGR are the principal parameters that must be transmitted. The following step consists in the grounding stage. Generally speaking, the grounding is considered as a mapping from an abstract to a concrete specification of the service description elements [20][21]. The Web Services Description Language (WSDL) is used as an initial grounding mechanism for OWL-S [20][21][22]. At this point we create an instance of *wsdlAtomicProcessGrounding* for each atomic process that was created before. In order to link the profile, process and grounding we have to assign those instances to the appropriate properties (see Figure.12).

In the light of these inputs and outputs parameters passing, the system can infer the list of user requirements at each point of functioning. Special emphasis is given in this paper for the graphical and functional representation of an elementary action and user's activity. Formal description of these models, then, the theoretical representation of the dynamic composition rules will be given in next papers.

 ISBN: 978-1-61208-171-7

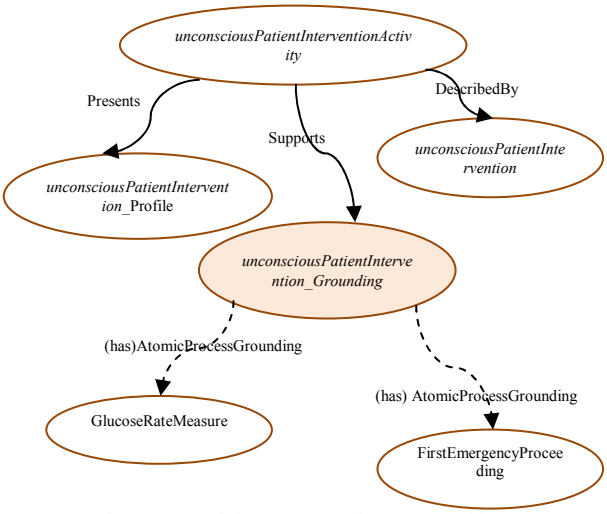

Figure 12. Activity representation expressed in OWL-S

IV. CONCLUSION AND FUTURE WORK

The fundamental goal of the interface adaptation to dynamic context of use in ubiquitous environments is to provide the relevant information to the user at the appropriate moment. Deducing the user's requirements in terms of information at runtime can arguably contribute to improve the suppleness of the user interfaces. This paper introduced a realtime modeling approach of the user's interface. Petri-nets technology was used to formulate the user's behavior and therefore infer the list of user's requirements at each point of functioning. The central goal of our work is to give an innovative approach for a better deduction of the user's requirements in a ubiquitous environment. In fact, a fixed model cannot adequately reach such objectives. To address this problem the presented approach enhances the adaptation process while improving models at runtime; we deal with evolutionary and dynamic models. Such models can be created while integrating progressively a range of elementary actions or undergo modifications and changes as the result of interactions with the user and through reinterpretations of existing models stored by the acquisition of preceding knowledge. Our approach takes advantage of OWL-S's properties in order to describe the dynamic functioning of Petri-nets models. We formulate a Petri-nets based elementary action by using an OWL-S atomic process. And then, we progressively compose a set of elementary actions to formulate a Petri-nets based activity. The presented method lays a sound foundation for dynamic composition of Petri-nets based modeling. As future work, a formal specification of the dynamic composition rules will be studied.

REFERENCES

[1] Víctor López-Jaquero, Jean Vanderdonckt, Francisco Montero and Pascual González. Towards an Extended Model of User Interface Adaptation: The Isatine Framework. Computer Science, 2008, Volume 4940/2008, pp 374-392.

[2] John Krumm (2010).Ubiquitous Computing Fundamentals. Redmond, Washington, U.S.A. 2010 by Taylor and Francis Group, LLC.ISBN 978-1-4200-9360-5.

[3] Grzegorz Lehmann. Runtime Models for Ubiquitous User Interfaces. W3C Workshop on Future Standards for Model-Based User Interfaces, May 13-14th, 2010, Rome, Italy.

[4] Sina Golesorkhi. Context Aware Dynamic Adaptation and Optimization of Web User Interfaces. Thesis's memory. November 2010. Rheinische Friedrich-Wilhelms-Universität Bonn - Institut für Informatik III.

[5] Nezhad, Hamid Reza Motahari, Xu, Guang Yuan and Benatallah, Boualem (2010): Protocol-aware matching of web service interfaces for adapter development. In: Proceedings of the 2010 International Conference on the World Wide Web 2010. pp. 731-740

[6] Víctor López-Jaquero, Jean Vanderdonckt, Francisco Montero and Pascual González. Towards an Extended Model of User Interface Adaptation: The Isatine Framework. Computer Science, 2008, Volume 4940/2008, pp 374-392.

[7] Myers B.A. (1995). User interface software tools. ACM Transactions on Computer-Human Interaction, 2 (1), pp. 64-103, March.

[8] Javier Criado, Cristina Vicente-Chicote, Nicolas Padilla and Luis Iribarne. A Model-Driven Approach to Graphical User Interface Runtime Adaptation. 5th Workshop on Models@run.time at MODELS 2010.

[9] Jacob R.J.K. (1986). A specification language for direct-manipulation user interfaces. ACM Transactions on Graphics, 5 (4), pp. 283-317, October.

[10] Olsen D.R. (1983). MIKE: the Menu Interaction Kontrol Environment. ACM Transactions on Information systems, 5 (4), pp. 318-344.

[11] Singh G. & Green M. (1991). Automating the lexical and syntactic design of graphical user interfaces: the Uofa* UIMS. ACM Transactions on Graphics, 10 (3), pp. 213-254, July.

[12] Ismail I. and Moussa F. « User Requirements Deduction in a Pervasive Environment». NGMAST: IEEE International Conference on Next Generation Mobile Application, Services and Technologies. Juillet 2010.

[13] F. Moussa, M. Riahi, C. Kolski and M. Moalla. Interpreted Petri Nets used for Human-Machine Dialogue Specification in Process Control: principles and application to the Ergo-Conceptor+ tool. Integrated Computer-Aided Engineering, 9, pp. 87-98, 2002.

[14] Riahi, M., & Moussa, F., (2001). Contribution of the Petri Nets and the multi Agent system in HCI Specification. 9th International Conference on Human-Computer Interaction. New Orleans Fairmont. Louisiane.

[15] Moussa, F., Kolski, C., Riahi, M. A model based approach to semiautomated user interface generation for process control interactive applications. Interacting with Computers, 12, pp. 279-292, 2000.

[16] OWL-S: Semantic Markup for Web Services, available at: http://www.w3.org/Submission/OWL-S/. Last update 22 November 2004. Last consultation May 2011.

[17] PHAN Quang Trung Tien. Ontologies et Web Services. Activity Report. Institut de la Francophonie pour l'Informatique. Hanoï, juillet 2005.

[18] Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001 Latest version: http://www.w3.org/TR/wsdl Erik Christensen, Francisco Curbera, Greg Meredith and Sanjiva Weerawarana Heidelberg. Last consultation May 2011.

[19] N. Khelil. Man-Machine Interface modeling. Master's memory, University of Tunis, october, 2001.

[20] R. Akkiraju, J. Farrell, J.Miller, M. Nagarajan, M. Schmidt, A. Sheth, K. Verma, "Web Service Semantics - WSDL-S," UGA-IBM Technical Note, version 1.0, April 18, 2005. http://lsdis.cs.uga.edu/ projects/METEOR-S/WSDL-S. Last consultation May 2011.

[21] Nezhad, Hamid Reza Motahari, Xu, Guang Yuan and Benatallah, Boualem (2010): Protocol-aware matching of web service interfaces for adapter development. In: Proceedings of the 2010 International Conference on the World Wide Web 2010. pp. 731-740

[22] Duy-Ngan Le; Van-Quoc Nguyen; Goh, A.; Matching WSDL and OWL-S Web Services. IEEE International Conference on Semantic Computing, 2009. ICSC '09. 14-16 Sept. 2009.Berkeley, CA.

 ISBN: 978-1-61208-171-7