# DYNAMIC AND REALTIME MODELLING OF UBIQUITOUS INTERACTION

Imen Ismail and Faouzi Moussa

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

**ABSTRACT**

*Ubiquitous systems require user to be dynamically and realtime informed in order to make his current activity increasingly easy. First, this paper presents and discusses a method to model the realtime interaction of the user with a ubiquitous system based on Petri-nets modelling technology. The goal deals with investigating dynamically the appropriate form of interaction depending on the context of the user. Thus, the interaction model structure should be dynamically improved with respect to the current and particular activity or goal of the user to better cope with his runtime requirements. This mechanism has been characterized as "models mutation". Secondly, this paper proves the dynamic construction of models while basing on the dynamic composition of services. The ultimate purpose is to take advantage of the ontology of service written in OWL-S in order to describe the dynamic aspect of Petri-nets based models, especially, the realtime and automatic composition of such models. Simulation work has been conducted to validate the proposed approach.*

**KEYWORDS**

*Petri-nets; Modelling; OWL-S; Model Mutation; Ubiquitous Interaction; Ubiquitous Environments*

## 1. INTRODUCTION

Man-System Interaction with increasingly more complex environments is a major concern. This paradigm is becoming more and more challenging with the emergence of the smart and ubiquitous environments leading to a new form of interaction i.e. the ubiquitous interaction. One of these challenges is to know what information is the most appropriate for the user, when this information will be needed and how it should be provided [1]. A system which reacts at runtime as expected by the end-user contributes to the reliability of the Man-System Interaction [2]. This paper presents and discusses a method to model the realtime interaction based on dynamic created models.

To address this problem, the proposed strategies will be based on evolutionary models and applies the Petri-nets technology to model the user's interaction. Such models can be created while integrating progressively a range of elementary models 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; this was called models mutation. The proposed approach takes advantage of the service ontology written in OWL-S in order to describe the dynamic aspect of Petri-nets based models, specially the realtime and automatic composition of such models. In short, a Petri-nets based elementary model is described by using an OWL-S atomic service. Then, progressively a set of elementary actions will be composed to construct the entire Petri-nets model.

## 2. THE UBIQUITOUS COMPUTING

The ubiquitous computing is a recent area of research considered as the new generation of Human Computer Interaction. The basic concepts of this new paradigm were founded in 1993 by Mark Weiser in his famous paper "*Some Computer Science Issues in Ubiquitous Computing*" [3]. In addition to computer science and engineering, the ubiquitous computing encompasses a wide range of other research subjects such as mobile computing, wireless network, artificial intelligence, as well as social and psychological sciences. This new technology deals with incorporating various electronic devices throughout our world. It consists in hiding these devices into our physical environment and our daily work [4][5]. The main purpose is to assist people in their daily activities without being noticed. Ubiquitous environment is defined as any physical space in which we integrate an important number of heterogeneous and smart devices, making a growing ubiquitous information network. Therefore, in such space, user is in front of enormous amount of information and heterogeneous content. Thus, he could see on his interface varied information. However, at some moment, user could need certain information and find the others completely useless: this choice depends necessarily on his/her preferences and his/her individual requirements.

### 2.1 Ubiquitous Systems

As previously said, the ubiquitous environments are highly dynamic environments where the context tends to evolve at runtime. Ubiquitous systems are systems that must work efficiently in such environments while providing the suitable information relevant to the interaction between a user and these systems. They will be able to adapt dynamically their behaviours based on the context change of users. This ability consists in perceiving the environment and detecting its contextual parameters. On the basis of these factors the system can react to the environment variations and responds to the user's requirements. The most oft-cited and widely accepted definition is given by Dey [6]: "*A system is context-sensitive (i.e. ubiquitous system) if it uses context to provide information and services relevant to the user, where relevancy depends on the task requested by the user*". For instance, a mobile phone with this ability may know that it is currently in the meeting room, and that its owner has sat down; in addition, it may conclude that the user is currently in a meeting and reject any unimportant calls.

### 2.2 Overview of the Proposed Ubiquitous System

The key goal of the proposed system [7] consists mainly in developing a formal modelling of the user's behaviour and applies specific processing to that model in order to identify the realtime required information. In light of this fact, the system tries to build the user's interface on an as needed basis; i.e. by mapping the best possible interface components (so-called widgets) with this information to concretely perform the target interface [7]. To overcome this purpose, the retrieval mechanism is built on a realtime and dynamic user activity modelling. Thus, the system could have a suitable representation of the user in a particular situation on realtime. Hence, it can deduce the suitable model or build a realtime model according to the analysis of the user-system interaction and the current context of use. It can then adaptively offer user information concerning what he/she is doing.

The proposed system comprises three functional phases (Figure 1) which are described in more details in [7]. The contextual data processing is the primary phase. In fact, pervasive environment is a greatly dynamic environment where context can evolve continually and at runtime. Therefore, an analysis step of this environment is necessary. The output is a document describing the contextual information. It represents the context model that will be used later by the adaptation strategy module. The gathered data which represents the context need to be stored in a suitable form. By analyzing related work on contextual data modelling, we concluded that ontology based languages seems to be the most appropriate candidate model [8]. In our context model, data are expressed by ontology written in OWL (Ontology Web Language).

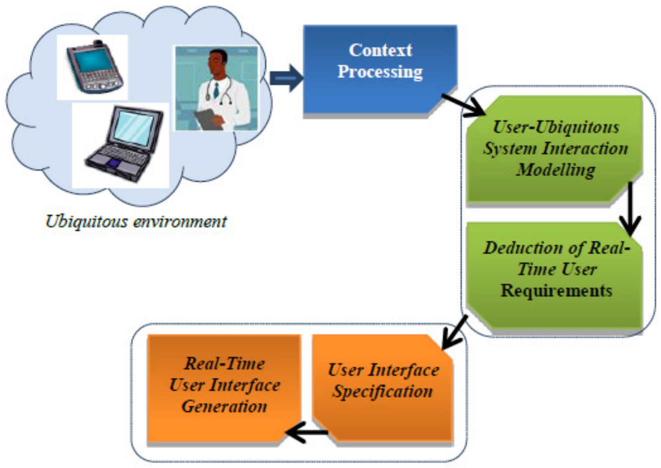

Ubiquitous environment
Context Processing
User-Ubiquitous System Interaction Modelling
Deduction of Real-Time User Requirements
User Interface Specification
Real-Time User Interface Generation

Figure1. The ubiquitous system functioning stages

The second phase concerns the adaptation strategy. It is divided into two modules:

*User-Ubiquitous System Interaction Modelling*: The user's activity must be analyzed in this second step, which will be further discussed later. A user's activity is composed of a set of elementary actions. The elementary actions were, first, modelled by elementary structures of Interpreted Petri-nets. Then, we proceed by typical compositions as sequential, parallel, choice, iteration, etc. A global model is then constructed integrating the different evolutions of the ubiquitous system operation. Likewise, the user's behaviour is modelled. It expresses the interaction of the user with the graphical interface. In this way, the user-system interaction model is achieved. It allows, afterward, the deduction of the user requirements.

*Deduction of User requirements and Identification of the required Widgets*: The next step ensures the identification of the user's requirements in terms of information in a first stage, and in term of interface widgets in a second stage.

The last phase consists of two steps that describe the interface generation. Once the interface widgets have been identified in the previous phase, this step consists in specifying the interface in terms of displays and graphical objects: this is the user interface specification phase. The final step represents the automatic and realtime user interface generation.

## 3. Interpreted Petri Nets for modelling the User's Interaction with Ubiquitous Environments

### 3.1 The Interpreted Petri-nets

The Petri-nets technology is a formal and abstract method to specify and to model a data flow. It was proposed by Carl Adam Petri in 1962 [9]. Generally, this method is showing an efficient and accurate way for specifying and describing the dynamic behaviour of various systems. It includes especially the distributed modelling of parallelism and synchronization in such systems. In particular, Petri-nets are continuously expanding and represent an appropriate tool for modelling the human-Machine Interaction [10]. At first they were used only for tasks description. But later, especially with the emergence of High-Level Petri-nets, modelling human-machine dialogue based on Petri-nets' profits has been started. We included principally Willem and Biljon's works. They were the first to model Human-Machine dialogue based on Petri-nets [11].

Petri nets are a graphical method for the modelling. They are in the form of bipartite graphs based on two types of nodes:

○ Represents Places (*Conditions*)
▭ Represents Transitions (*Discrete Event may occur*)

These nodes are connected by arrows that indicate the direction of the data flow and describe what places are pre-and / or post-conditions for each transition (Figure 2).
A Petri-net is a tuple *(P, T, Pre, Post)* where:

*P = {p1, p2, p3, ..., Pn} is the set of places*
*T = {t1, t2, t3, ..., tn} is the set of transitions*

• **Input function**: Pre: P x T $\rightarrow$ N, Defines the necessary number of tokens in place $P_i$ (input) of a transition $T_j$ enabling this transition.
• **Output function** Post: P x T $\rightarrow$ N, Defines the number of tokens necessary for the place $P_i$ (output) of a transition $T_j$ after firing this transition.

Petri-nets represent statements or resources in the system while transitions model the system activities. With the emergence of the high-level Petri-nets [12], the study of a wide range of applications becomes easier. These new models were mainly based on the basic concepts of Petri nets; the initial model was enriched by extensions and a variant of Petri nets models. Among these models we quote: the Coloured Petri-nets, the Synchronized Petri-nets, the Synchronized and Timed Petri-nets, Stochastic Petri-nets and Interpreted Petri-nets.

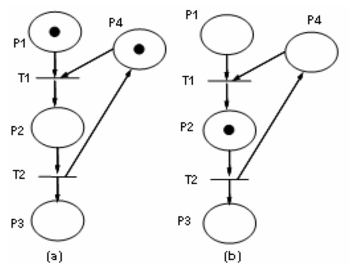


Figure 2. Petri-nets graphical Example before and after firing transition

The Interpreted Petri-nets will be defined by the set [13]:
*< P, T, E, W, Pre, Post, $\mu$, Precond, Action, Info-Transition >*
where:

- ***P*** : set of places = $\{P_1, P_2,..., P_n\}$,
- ***T*** : set of transitions = $\{T_1, T_2,..., T_m\}$,
- ***E*** : set of events including the event "always present" <e>,
- ***W*** : set of the interface widgets,
- ***Pre***: P x T $\rightarrow$ N defines the weight of the bow joining a place $p_i$ of P to a transition $t_k$ of T,
- ***Post***: P x T $\rightarrow$ N defines the weight of the bow joining a transition $t_k$ of T to a place $p_i$ of P,

- ***μ***: T → E associates to each transition the appropriate triggering event,
- ***Precond***: T → Boolean Expression defines the necessary passing condition for each transition,
- ***Action***: T → A defines the eventual and appropriate action procedure associated to each transition
- ***Info-Transition***: T → W associates to each transition, the appropriate widget(s).

## 3.2 Interpreted Petri-nets for Human-System Interaction Modelling

Petri-nets were chosen as a formal technique for the interaction modelling. The essential reason behind this choice is that the Petri-nets based models can reliably describe the aspects of concurrency, parallelism and dynamism which are fundamental features of ubiquitous environment. This extension introduces the notion of events and conditions as well as the notion of actions. We associate a passing condition ($Cond_j$), a triggering event ($Ev_j$) and a possible action ($Action_j$) to each transition ($T_j$) (Figure 3). To model the user's behaviour using interpreted Petri-nets the places represent the user's behaviour according to the system's evolution, in particular the environment evolution and changes.

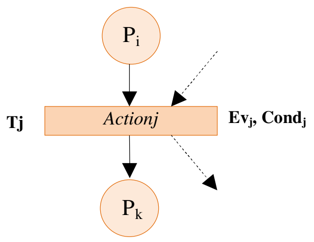


Figure 3. Interpreted Petri-nets modelling

The user's activity model is composed of a set of elementary action's sub-models. Figure 4 depicts an example of elementary action representation, where the places represent the user's interaction according to the system's evolution and changes. In particular, the places $P_{ai}$ and $P_{ci}$ model the state of the user before and after the execution of the action; whereas the place $P_{bi}$ describes a waiting state. Two main types of events trigger the end of the elementary action and indicate that it has been successfully executed: The validation of the condition *i* as well as the presence of the event "End action". It is the user-interaction model that we wish to describe. This model is built by composing all the required elementary actions. As per the requirements, these elementary actions are scheduled according to typical compositions as sequential, parallel, choice, iteration, etc.

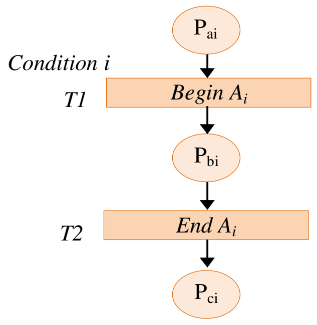


Figure 4. Elementary activity modelling

The sequential composition of N elementary actions is done by merging the output places of the Petri-nets that describes the first action ($A_i$) and the input places of the next action ($A_j$). It should be noted that this actions' sequence is supposed to be possible. Figure 5 shows an example of sequential composition.

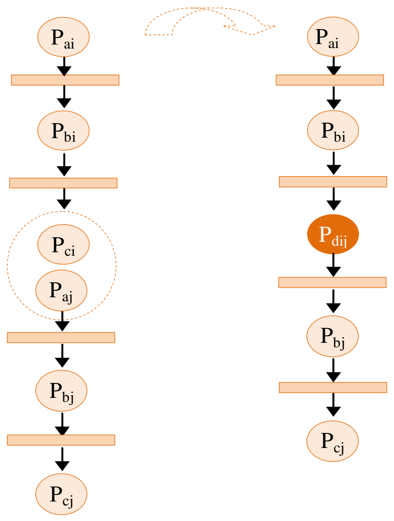


Figure 5. Sequential composition of two elementary actions

The parallel composition enables two or more actions' execution in a simultaneous manner (Figure 6). An input place, called synchronization place, must at the same time activate all initial places of the parallel actions. The parallel composition of *n* Petri-nets based-model actions involves two particular composition structures: parallel structure 1 and 2 (resp. PS1 and PS2). PS1 represents the starting synchronization of the *n* Petri-nets models (Figure 7a) whereas the final synchronization of these models is described by the PS2 structure (Figure 7b).

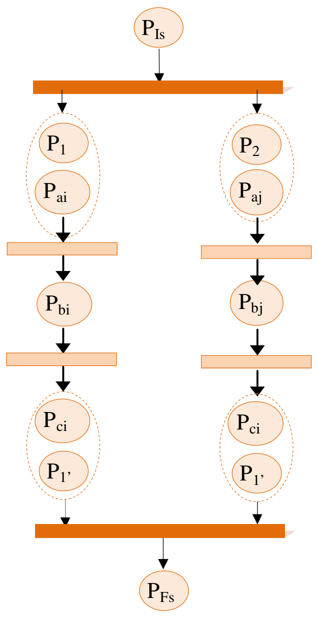


Figure 6. Parallel composition of two elementary actions

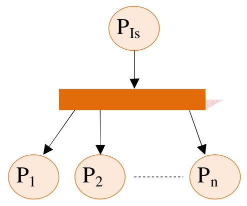


Figure 7. (a) PS1 structure

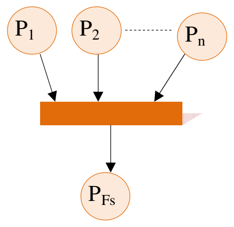


Figure 7. (b) PS2 structure

A global model is then constructed by integrating the different evolutions of the functioning states of the ubiquitous system. Likewise, the user's interaction is modelled. Once the user's behaviour has been modelled, the user requirements can be deduced. In fact, we consider a transition "Begin Action" in the user's behaviour model. We associated the adequate parameter(s) to these transitions. These parameters refer to the information required at this working point within these specified operating conditions. Once these parameters have been identified, we can deduce the necessary components and widgets of the user interface [7].

We consider the detection, the evaluation of the situation, the decision, and the action. The evolutions between these states are modelled by the transitions. It should be noticed here that any change of the environment state may affect the user interface layer. In other words, it will imply a change in the graphical display affecting the widget's parameters or the display contents (appearance or disappearance of some widgets, etc.). This modelling technique allows us to deduce on realtime the user requirements according to the current context of use, including the user's activity.

## 3.3 Motivating Example

Ambient intelligence and ubiquitous computing have large advanced application areas. Computer science technology applied to personalized medicines benefits greatly from research done in this area. This subsection presents a case study in the recovery room of a hospital, variously called a post-anaesthesia care unit. It's a space a patient is taken to after surgery to safely regain consciousness from anaesthesia and receive appropriate post-operative care. To achieve such objectives, the nurse (or any medical staff) may do several activities. Two specific activities were considered to illustrate what can happen for the system functioning when a user must suddenly change activities. The first activity is to carefully monitor patient's state evolution right after surgery. The second one deals with an immediate intervention when a specific type of sensor, a pulse *Oximeter*, warned a very low level of oxygen saturation, indicating consequently a priority activity to be undertaken. Regarding the former situation, two elementary actions (*Ea1* and *Ea2*) will have to be conducted.

A set of key information must be provided to the nurse in order to successfully perform his current activity. First, for *Ea1's* accomplishment (i.e. to update the patient's chart) the nurse needs to know patient's data such as: the responsible surgeon information, the time of the operation, the operation room (room number, floor and service) and some information about the anaesthesia phase (anaesthetist's details, estimated duration before recovery). Then, the nurse must monitor carefully (*Ea2*) vital signs (e.g. Pulse, blood pressure, temperature, blood oxygen levels…) as the effects of anaesthesia wear off. The model depicted by Figure 8 should govern the system functioning to address this interaction and adequately deduce all these aforementioned data at the appropriate moment.

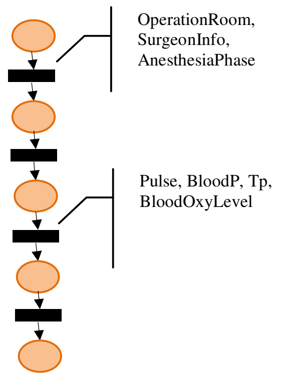


Figure 8. Interpreted Petri-nets Model of the first activity (*Ea1* ***Seq*** *Ea2*)

Suppose the system detects a very low OSL threshold related to another patient in the recovery room when no other nurses or doctors are available. This patient is in danger of losing his life unless an urgent intervention has to be carried out. Dealing with this case is, therefore, seen as a priority for the nurse. The Petri-nets model which represents this new activity is depicted by the Figure 9. A nurse must perform two parallel elementary actions (for example namely Ea6 and Ea7). The first step is to administrate supplemental oxygen through a nasal cannula or face mask. The second action consists in administering an intravenous fluid to the patient. Nevertheless, given the sensitive nature of this action, it must be carried out by a specialist nurse.

Another question is how the system should support the nurse when it discovered that he is not specialist. Since any decision must be in keeping with the responsible doctor-surgeon, the nurse needs to know the doctor and his contact details. An elementary action (Ea14) enables to meet such objectives (Figure 10). This action requires the patient's identity as input and provides the doctor's general details as outcomes. Implicitly, it gives the responsible doctor-surgeon details in the case of recovery room (i.e. according to the user's places).

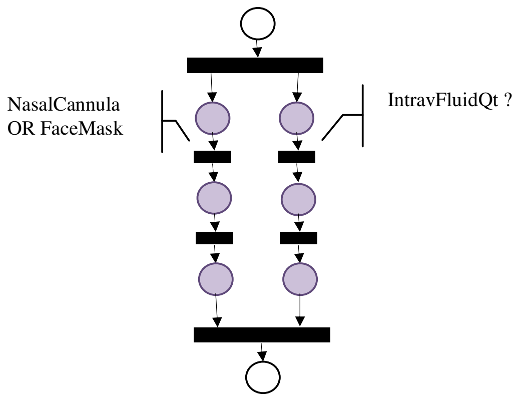


Figure 9. Interpreted Petri-nets model of the second activity (*Ea6* ***Par*** *Ea7*)

These, in short, are the different models which should govern the nurse's intervention faced with such particular conditions and circumstances and with respect to a given composition. The suitable interpreted Petri-nets model to cope with the previous described situation is depicted by Figure 11. This means that the execution of the *PatientStatusEvaluation* process should be interrupted and turns to a sequential execution of the second process model, i.e. *AdministrateOxygen*. As well, based on the nurse's information, which has been obtained from his online, OWL-encoded profile, the entire model should be improved by the parallel execution of the elementary action *CallDoctor*. Finally, the execution of the last elementary action will be performed.

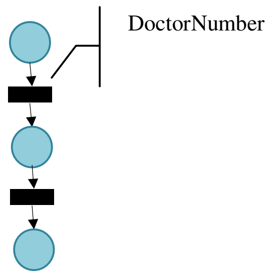


Figure 10. Interpreted Petri-nets model of the *Ea14* activity

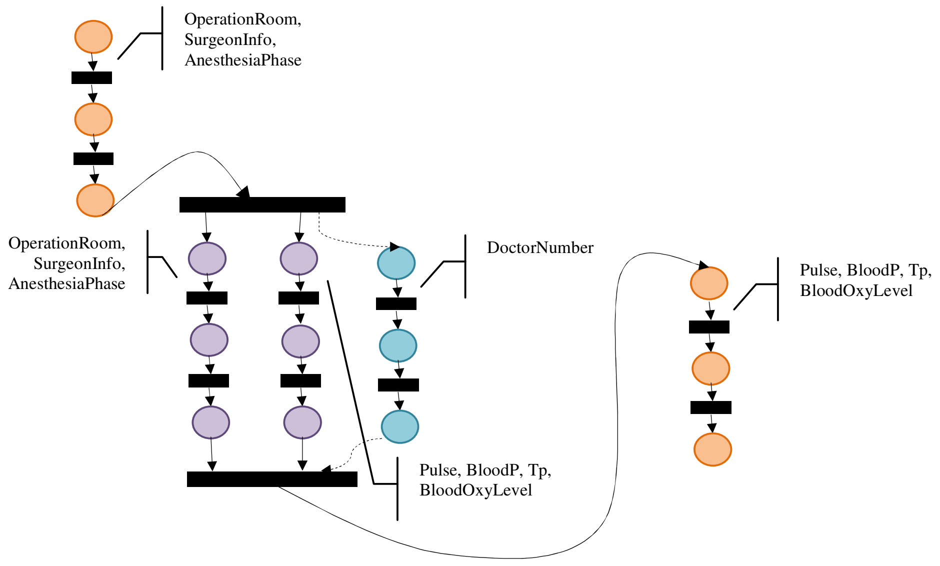


Figure 11. Suitable Petri-nets based model describing the current situation

## 4. Implementation of Interpreted Petri-nets models mutation based on OWL-S specifications

The proposed approach mainly refers to the variation of the user's activities. Obviously, each activity may apply for specific information in order to be accomplished. Thus, the models, on which the system's operation is based to deduce these specific data, should be built in proportion as runtime evolving of ubiquitous environment as well as according to available information. This means that the model can progress and change its structure, so the system can provide for adequate controls over the management of the user-system interaction. Within this context, the mutation of Petri-nets based modelling is addressed. Taking into account that our ontology is described by OWL [14], the OWL-S [15] technology best fits our requirements. With OWL-S as starting point we are going to describe the Interpreted Petri-nets model's mutation.

### 4.1 Modelling Services based on OWL-S Specifications

Services or more precisely, Web services are software components that communicate using pervasive, standards-based Web technologies including HTTP and XML-based messaging. They are designed to be accessed by other applications and vary in complexity from simple operations to complex processes running. The main benefit of this technology lies in the fact that it establishes a common, vendor-neutral platform for integrating distributed computing applications. Semantic Web Services promise to provide solutions to the challenges associated with automated discovery, dynamic composition, enactment, and other tasks associated with managing and using service-based systems [16]. OWL-S is OWL-based web service ontology. It supplies a core set of markup language constructs for describing web service in computer-interpretable form [15]. Each service is characterized by three main concepts: Profile, Process and Grounding (Figure 12).

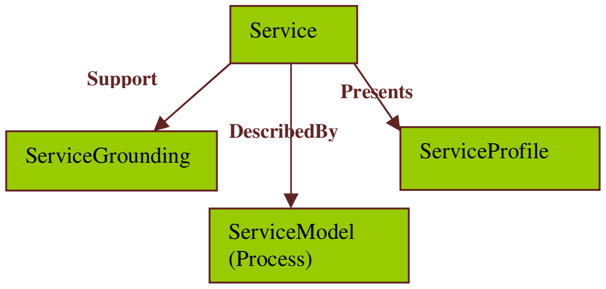


Figure 12. Representation of the service structure

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]. The Grounding specifies on how to interoperate with a service via messages [18]. There are three key properties that describe a service profile. *Service Name* which refers to the name of the service, it can be used as an identifier. The *Text Description* which provides a brief description of the service, including what the service requires to work. The *Contact Information* that provides a mechanism of referring to humans or individuals responsible for the service.

An OWL-S process is based on an IOPR model (Inputs, Outputs, Preconditions and Results). The Inputs are information required for the execution of the service, whereas outputs are information that the process returns to the requester. The *preconditions* are determining factors imposed over the inputs and that must hold for the process to be successfully invoked. The *result* entity specifies means to meet the process’s results with corresponding outputs. Each result can be associated to a result condition, called inCondition, that specifies when that particular result can occur. Therefore, an inCondition binds inputs to the corresponding outputs. It is assumed that such conditions are mutually exclusive, so that only one result can be obtained for each possible situation.When an inCondition is satisfied, there are properties associated to this event that specify the corresponding output (withOutput property) and, possibly, the Effects (hasEffect properties) produced by the execution of the process.

The execution of a process may also result in changes of the state of the world. Note that there is a fundamental difference between effects and outputs. Effects describe conditions in the world, while outputs describe information. OWL-S does not assume that outputs and effects are the same for every execution of the process. Rather, it allows the specification of the set of conditions under which the outputs or the effects may result. Because of the need for conditions, OWL-S defines the classes of *ConditionalOuput* and *ConditionalEffect*. Both classes allow a number of conditions to be associated with the outputs and the effects respectively. Unconditional outputs and effects are defined by leaving the list of conditions empty implicitly saying that the condition is always true.

Two classes of services can be identified depending on the complexity of the interaction with a service: Atomic Services and Composite Services [15]. 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. 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 involved services. The atomic process is a description of an atomic service which can be directly called while passing appropriate messages. The composite process is decomposable into atomic processes or other composite ones while using control constructs such as sequence, split, choice, if-then-else, iterate, repeat-while, and repeat-until.

## 4.2 Interpreted Petri Nets modelling based on OWL-S processes

This subsection explains our approach for enabling an interpreted Petri-nets based model to be dynamically executed through the OWL-S specification and properties. The goal is to improve a method so as to adapt user-system interaction successfully and dynamically into the contingent needs to achieve a sufficient fit between user's requirements and the perpetual change of his/her environment. The key idea is to formulate a Petri-nets based elementary action by using an OWL-S atomic process, and formulate a Petri-nets based activity while composing progressively a set of atomic processes. The idea is based on the hypothesis that the elementary action is a non-decomposable action. The activity is composed of elementary or composite actions through well-defined composition rules. These rules govern the ordering and conditional execution of the action in the model.

### 4.2.1 Encoding IPN-based Elementary action as OWL-S Atomic Process

The basic concept of the approach is to encapsulate an elementary action by an atomic process. Therefore, the elementary action's parameters can be specified by the IOPR model of an atomic process. In particular, the input parameters are information needed for the well accomplishing of the action. The output information is information provided by the action. The precondition parameters are the condition imposed over the inputs of the action to be successfully executed. Local parameters are only used in atomic processes. These essential parameters are summarized in the following table (Table1). Some other non-functional properties can be considered including local parameters, name, category and goal [15].

Table 1. Fit between OWL-S based Process's properties and Elementary action Parameters

| **Elementary action Parameters** | **OWL-S based Process's properties** | **Type** |
|---|---|---|
| Condition i | hasPrecondition | EXPRESSION (KIF-Condition) |
| BeginAi | hasInput | PARAMETER : TRNSITION |
| Pai | hasLocal | PARAMETER : PLACE |
| EndAi | hasOutput | PARAMETER : TRANSITION |
| Pci | hasLocal | PARAMETER : PLACE |
| Requirements | hasResult | REQUIREMENTS_LIST |
| ActionName | Name | STRING |
| Pbi | hasLocal | PARAMETER : PLACE |

The following OWL-S code gives an extract specification of the elementary action *Ea1* depicted by Figure 13 For the effective beginning, this action needs *Pai* and *BeginEa1* as inputs, as well as the patient identification (*PatientId*) as **precondition on result**. *EndEa1* and *Pc1* represent the outputs of the action. Ea1 should provide as results the post-operative information of the given patient; more precisely, properly tests for the existence of the patient in the database and conducting research on patient's information. Results include *OperationRoom*, *SurgeonInfo*, *AnesthesiaPhase* which represent the nurse's needed information to satisfactorily perform his current activity.

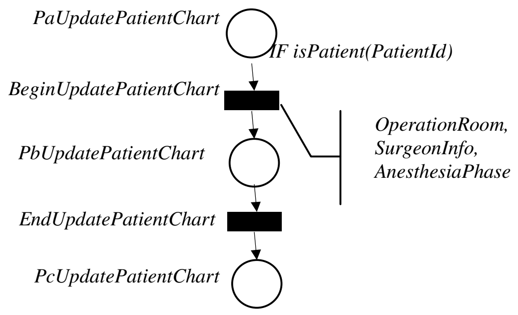


Figure 13. Simplified representation of the elementary action Ea1

```
<!— Ea1: Service description -->
<service:Service rdf:ID="UpdatePatientChartService">
<service:presents rdf:resource="#UpdatePatientChartProfile"/>
<service:describedBy rdf:resource="#UpdatePatientChart"/>
<service:supports rdf:resource ="#UpdatePatientChartGrounding"/>
</service:Service>
<process:AtomicProcess rdf:ID="UpdatePatientChart">
      <process:hasInput>
            <process:Input rdf:ID="PaUpdatePatientChart"/>
      </process:hasInput>
      <process:hasInput>
            <process:Input rdf:ID="BeginUpdatePatientChart"/>
      </process:hasInput>
      <process:hasOutput>
            <process:Output rdf:ID="EndUpdatePatientChart"/>
      </process:hasOutput>
      <process:hasOutput>
            <process:Output rdf:ID="PcUpdatePatientChart"/>
      </process:hasOutput>
<process:hasPrecondition>
      <expr:SWRL-Condition rdf:about="#SWRL-IsPatient()">
            <expr:expressionObject>
            <swrl:AtomList rdf:ID="AtomListID2"/>
            </expr:expressionObject>
      </expr:SWRL-Condition>
</process:hasPrecondition>
<process:hasLocal>
      <process:Local rdf:ID="PbUpdatePatientChart"/>
</process:hasLocal>
<process:hasResult>
      <process:Result rdf:ID="OperationRoom">
            <process:hasResultVar>
                  <process:ResultVar rdf:ID="VarOperationRoom">
                        <process:parameterValue rdf:parseType="Literal">
                        </process:parameterValue>
                  </process:ResultVar>
            </process:hasResultVar>
      </process:Result>
```

```
</process:hasResult>
(…)
</process:AtomicProcess>
```

### 4.2.2 Encoding IPN-based Activity as OWL-S Composite Process

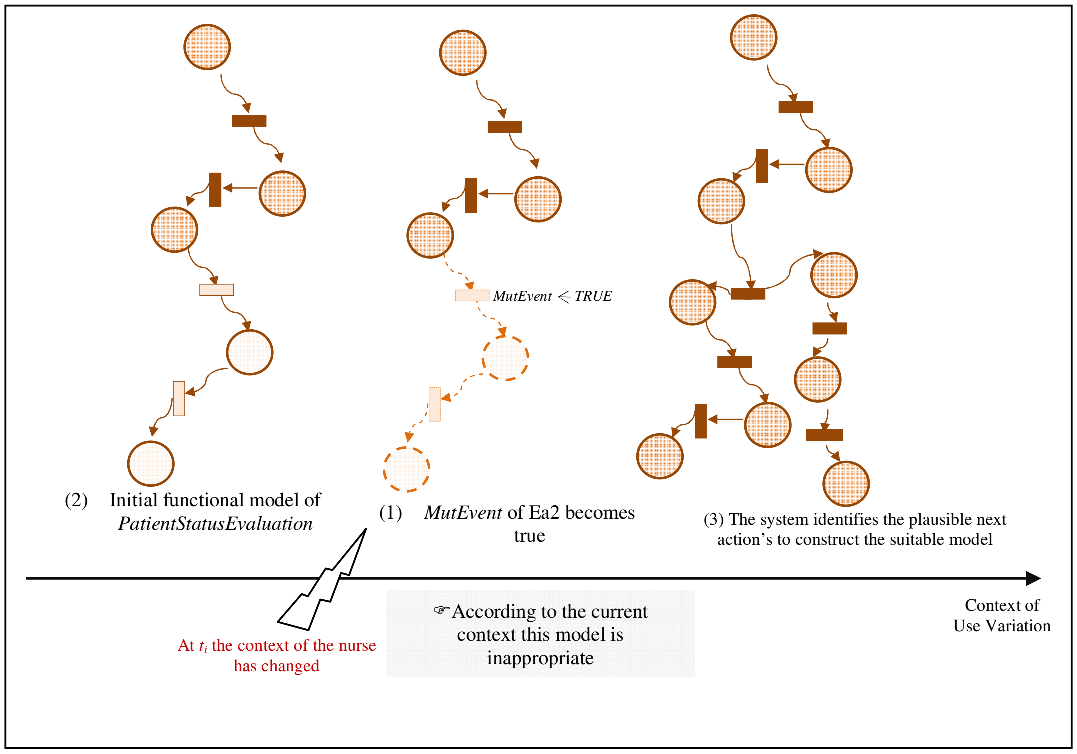
As previously noted, an activity is an aggregation of elementary actions scheduled according to typical compositions as sequential, parallel, choice, iteration. Developing the runtime model of the user's activity is based on operational compositions of elementary actions models. Since an elementary action is encapsulated by an atomic process, thus an activity can be modelled by a composite process mainly thanks to the control constructs which enable the composition of elementary actions. Examples of these constructs include sequence control which allows a list of processes to be done in order; the split control executes a number of processes concurrently; the execution of processes until the while Condition becomes true is done by the Iterates control, etc [15]. As an illustration, take the example of the first activity, called "*PatientStatus-Evaluation*". This activity is composed of two sequential elementary actions, i.e. *Ea1* and *Ea2*. Ea1, *UpdatePatientChart*, consists in monitoring the patient's state evolution right after surgery in order to update the patient's chart. Next is *Ea2*, *EvaluateCriticalSign*, where the nurse must monitor carefully vital signs as the effects of anaesthesia wear off. The following is a simplified example that illustrates the description of this composite process through OWL-S specifications:

```
(…)
<process:CompositeProcess rdf:ID="PatientStatusEvaluation">
<rdfs:label>The process for monitoring patient state after surgery
</rdfs:label>
      <process:ComposedOf>
            <process:Sequence>
                  <process:components rdf:parseType="Collection">
                        <process:AtomicProcess
                  rdf:about="#UpdatePatientChart"/>
                  <process:AtomicProcess rdf:about=
                   "#EvaluateCriticalSign"/>
                  </process:components>
            </process:Sequence>
      </process:ComposedOf>
</process:CompositeProcess>
(…)
```

## 4.3. Realtime and Dynamic Composition of Interpreted Petri-nets Model

Our aims are supposed to benefit from the specified features of OWL-S specification to dynamically create a model, notably the running, the dynamic composition, the invocation and others particular features. Within OWL-S specification, the execution of a process may result in the data transfer from inputs into outputs. This, while generating some new information based both on given data as well as the environment state. Such new information is specified by the **inputs** and **outputs** of the process. In addition, a process can produce a change of the state of the world. This transition is described by the **preconditions** and **effects** of the process: *Effects* are changes in the state of the world, and *Preconditions* specify conditions that should be satisfied for a process to be executed correctly.

The ubiquitous environment changes are one of the most important information sources that the system needs to have instantly in the course of its operation. In particular, events that are sources of urgent intervention of the system can be specified by the parameter *Effects*. Therefore, the carrying out of the model continues to be held as normal until a new event is triggered. The model must reform itself at that time according to the new situation. With reference to the approach above-described, the system should identify and assess the next action supposed to be executed. It shall also control the connection conditions to the current action. Here, in particular, the data flow specification must be defined as well as the nested control constructs from which it is composed, and their ordering too. More precisely, a precondition must be carried out for each *EndAction* transition of an elementary action. This precondition, called ***MutEvent*** will be represented by a Boolean variable which must be set to false in order to enable this transition. This value becomes true upon detection of a change event while the system is searching for the next necessary action to be executed. In the proposed example, the system initiates its operation by identifying the appropriate model required to perform the first activity: *PatientStatusEvaluation*. The conditioning change of the current context, semantically specified by the *MutEvent* parameter of the Ea2, makes the remainder of the current model no longer suitable. Thus, the system identifies the more suitable action according to the new situation (Figure 14).



MutEvent ⇐ TRUE
(2) Initial functional model of PatientStatusEvaluation
(1) MutEvent of Ea2 becomes true
(3) The system identifies the plausible next action's to construct the suitable model
At $t_i$ the context of the nurse has changed
☞According to the current context this model is inappropriate
Context of Use Variation

Figure 14. Model mutation according to the context change

## 5. Simulation and Results

For evaluation purposes, simulation work has been conducted to validate our approach. Based on OWL-S process execution, our aim is to simulate results while highlighting the models mutation compared to a normal execution of OWL-S processes based models. For these purposes, Live Sequence Charts (LSCs) tool [19][20] is used to simulate the processes communication. Furthermore, Play-Engine [21] is employed to perform automated visualization, simulation and checking processes execution. The main intention of LSCs and the Play-Engine tools is providing the means for monitoring scenario based behaviour in operation and direct testing of requirements [21]. Applied to OWL-S based process, LSCs is able to describe not only the way a service will be executed and invoked, but also the expected result and outputs.

Play-Engine supports also a mechanism that enable the specification, validation, analysis and execution of LSCs, called "play-in" and "play-out". Within LSC chart, two condition's types are distinguished: cold condition and hot one having different impacts on the process execution. The cold condition is used in complex control structure specification in particular do-while and choice. However, to assure critical properties at certain point of execution the hot condition is used. Thus, if any hot condition evaluated to false, a violation is caught, alternatively simulation continues with the user's provided events [21]. That way, a *precondition* of a process is identified with a **cold condition** at the beginning of the chart. If the condition is false the chart terminates and the process is not performed. The *inCondition* and *hasResult* properties are conjoined and identified with a hot condition. The *withoutput* property is identified with communication after the hot condition. An atomic process is described by a universal chart [20] represented as vertical lines; the messages between instances are represented through horizontal lines. About composite process, composition's properties are translated to the appropriate communication type. For instance, the sequence's property is translated to sequential communication along the vertical lines.

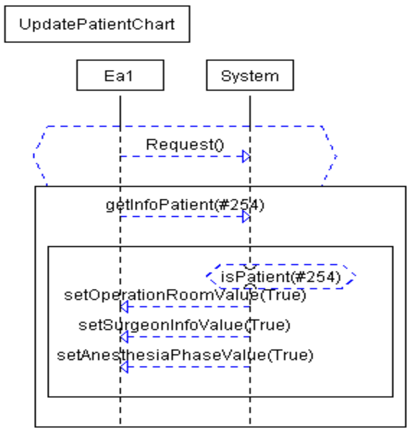
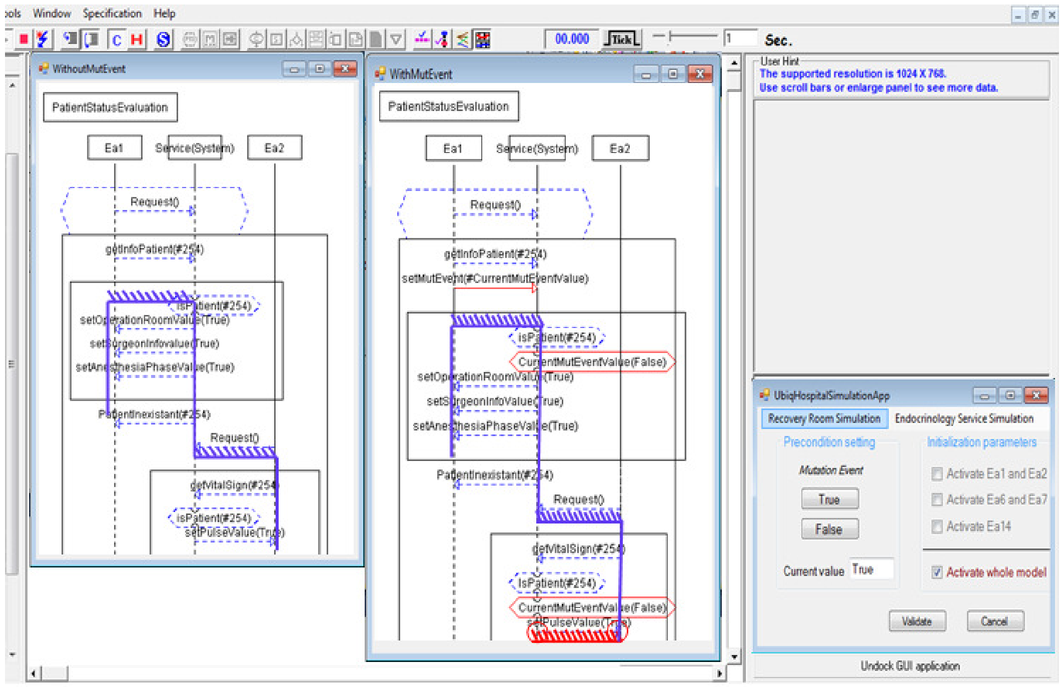

UpdatePatientChart
Ea1
System
Request()
getInfoPatient(#254)
isPatient(#254)
setOperationRoomValue(True)
setSurgeonInfoValue(True)
setAnesthesiaPhaseValue(True)

Figure 15. Scenario LSC to provide information about a given patient

Processes in Split correspond to multiple pre-service requests; each of them will activate an LSC modelling the corresponding service at the right moment. Further details about other properties are given in [17]. Figure 15 shows an LSC example that specifies the necessary interactions between the Ea1 process and one of the system's services *PatientInformation- ProviderService*. Once the *Ea1* process requests the system's service as well as if the value *PatientId* is validated, necessary information will be provided with True sign. In order to simulate process model operation interactively, a Play-Engine aware GUI application (Figure 16) is built enabling the manual events triggering; moreover it provides all the information required to ensure correct interaction with the Play-Engine.

Play-Engine supports such application type and plays-out the corresponding LSCs. Figure 16 shows the LSC of the *PatientStatusEvaluation* process model. Given the *PatientId* value as input and external event introduced through the *UbiqHospitalSimulationApp* application, the Play-Engine interface showcases how the system operates. The first diagram describes an operation deprived of the *MutEvent* precondition effects. Thus, the system continues functioning and along these lines it persists in providing information supposed to be inappropriate for the nurse. However, the second chart reveals how the system changes its current operation as a result of *MutEvent* triggering. Hence, the provided information is indeed the required one since the operating model is presumed to be the suitable model describing the current situation.



User Hint
The supported resolution is 1024 X 768.
Use scroll bars or enlarge panel to see more data.

Activate Ea1 and Ea2 [ ]
Activate Ea6 and Ea7 [ ]
Activate Ea14 [ ]
Current value True
Activate whole model [✓]

Figure 16. Comparison charts between normal functional modelling and dynamic functional modelling

## 6. Related Work

In this section, we are going to be talking about the operational applications of both Petri-nets and OWL-S concepts with a certain purpose. It should first be noted that the Petri-nets technology has been used generally as a formal analyzing mechanism and they are mainly used as a modelling tool, very efficient too [22]. However, combined together with OWL-S, the Petri-nets formalism provides an operational semantics to OWL-S since it defines formal and executable specifications to analyze, simulate check and validate OWL-S specifications [22][23]. In particular, Petri-nets are intended to analyze model of web services composition and calculate service performance. For example, [24] provides a translator from OWL-S description to Petri-nets that support formal analysis of OWL-S services thanks to Petri-nets benefits. In addition to performance analyzing, Petri-nets with objects (PNO) are used specifically for workflow service simulation and verification [19] while providing rules to derive OWL-S specifications from PNO specifications. Another example focuses on verifying the correctness of composite web services defined by OWL-S. The verification is mainly based on rigid mathematical ground of Petri-nets [25].

Another type of using Petri-nets within this context is given in [26], where a Petri net-based approach for modelling the choreography of OWL-S web services was described. Each control construct of the OWL-S choreography is represented through a Petri net pattern that captures formally its operational semantics. They also represent the flow of data, the outputs transformations, the effects in the environment, in addition to the structures that control the choreography of the services in their proposed Petri-nets models. However, to the best of our knowledge no approach in literature makes use of the OWL-S to model man-system interaction in ubiquitous environment in general, and modelling based on Petri-nets technology in particular. Our objective scrutinizes more on giving a new approach to construct dynamically and at runtime a Petri-nets based model toward deducing the user's requirements at the appropriate moment and according to his context of use. Many benefits of OWL-S allowed us to implement this modelling technique i.e. model mutation including the dynamic aspect of Petri-nets models. In addition, this enables to publish the composite activity's description thanks to OWL-S which is a semantic web service description language.

## 7. CONCLUSION

The very dynamic aspect of ubiquitous environments has significantly altered the nature of Human-System Interaction. Consequently, approaches of modelling these interactions have also to be changed. The aims are to reflect better and more efficient models that can cope with the dynamic aspect of such environments. A dynamic modelling approach is provided in this paper. The main purpose is to monitor and control the realtime interaction of the users while they remain in the ubiquitous environment. To make sure the users are provided with the most appropriate information at runtime is the main goal of dynamically modelling their behaviours. Effective implementation of this approach is mainly based on the web services technology specified in OWL-S language. More precisely, a Petri-nets based model that represents an elementary action is implemented through an atomic service. Then, we gradually compose a set of elementary actions to formulate the whole Petri-nets based model that represents the current activity. This mechanism is illustrated by a simulation tool suitable for the runtime visualization of OWL-S processes in operation. Particular emphasis of the dynamic alteration of the model to provide solution better suited to the real needs of the user is outlined. Future work will give further details on the automatic elementary action discovery and automatic invocation of services. Besides, future work will focus on implementing underlying data and functional system using techniques compatible with ActiveX DLLs, e.g. ASP, .NET, Play-Engine may import the actual implementation of the underlying system and perform the simulation. Formally, the work will concentrate on the invocation mechanism, i.e. service-using requests service-provider while determining whether the service meets its needs by exploring the service profile.

## REFERENCES

[1] Kranz, M., Holleis, P., Schmidt, A.: Embedded Interaction: Interacting with the Internet of Things. Internet Computing, IEEE, vol.14, no.2, pp.46-53 (2010)

[2] Tran, M.H., Han, J., Colman, A.: Social context: Supporting interaction awareness in ubiquitous environments. In Mobile and Ubiquitous Systems: Networking & Services, MobiQuitous, pp 1- 10 (2009)

[3] Mark Weiser, some computer science issues in ubiquitous computing. CACM, vol 36, issue 7, 1993.

[4] Noêlle Carbonell. Ambient Multimodality: an Asset for Developing Universal Access to the Information Society. International Conference on Universal Access in Human Computer Interaction (August 2007).

[5] G J Doherty, M D Wilson. Modelling Ubiquitous Computing Applications. In Proceedings of Workshop on Continuity in Future Computing Systems, Porto, Portugal, April 2001.

[6] A. Dey, D. Salber, and G. Abowd: "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications", Special issue on Context-Aware Computing in the Human-Computer Interaction (HCI) Journal, vol. 16 (2-4), pp. 97-166, 2001

[7] I. Ismail and F. Moussa: "A Pervasive System Architecture for Smart Environments". In International Journal of Artificial Intelligence & Applications (IJAIA), volume 3, Number 5 (pp113-126, September 2012

[8] I. Ismail: "Contextual Data Management in Ubiquitous Environment". International Conference on Computing and Information Technology. Saudi Arabia 12-14 March 2012.

[9] C. A. Petri. "Kommunikation mit Automaten". Bonn : Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962 ; New York : Griffiss Air Force Base, Technical Report RADC-TR-65--377, Vol. 1, pp. Suppl. 1. 1966.

[10] Matej, P., Matej, K., Janez, P., Gašper, M., Goran, V., Stanislav, K.: Analysis of multi-agent activity using Petri nets. In Pattern Recognition, 43 (4), p.1491-1501, (2010)

[11] R. Williem and V. Biljon. Extending Petri Nets for specifying Man-Machine dialogues, International journal of Man-Machine Studies, vol. 28, 1988, pp. 437-45.

[12] Decker, G., Weske, M.: Interaction-centric modelling of process choreographies. In Information Systems Volume 36, Issue 2, (Special Issue: Semantic Integration of Data, Multimedia, and Services), pp. 292–312 (2011)

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

[14] Owl2 the Web Ontology Language: Structural Specification and Functional-Style Syntax Boris Motik, Peter F. Patel-Schneider, Bijan Parsia, eds. W3C Recommendation, 27 October 2009, Latest version available at http://www.w3.org/ TR/owl2-syntax/. Last consultation March 2012

[15] Owl-s: Semantic Markup for Web Services, available at: http://www.w3.org/Submission/OWL-S/. Last update 22 November 2004. Last consultation May 2012

[16] Daniel Elenius, Grit Denker, David Martin, Fred Gilham, John Khouri, Shahin Sadaati, Rukman Senanayake. The OWL-S Editor - A Development Tool for Semantic Web Services. In ESWC, pp 78-92, 2005

[17] Quang, P., Tien, T.: Ontologies et Web Services. Activity Report. Institut de la Francophonie pour l'Informatique (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 2012

[19] Harel, D., Marelly R.: Specifying and Executing Behavioral Requirements: The Play In/Play-Out Approach. In Software and System Modelling 2, 82–107 (2003)

[20] Sun, J., Li, Y., Wang, H.H.: Visualizing and Simulating Semantic Web Services Ontologies. In ICFEM pp. 435-449 (2005)

[21] Ersoy, A. : An Evaluation of Live Sequence Charts and the Play-Engine. ; Master of Science Thesis, Stockholm, Sweden 2006. KTH Computer Science and Communication

[22] Khadka, R., Sapkota, B.: An Evaluation of Dynamic Web Service Composition Approaches. (2010)

[23] Alonso, G.: Web services: concepts, architectures and applications. Springer Verlag (2004)

[24] Antonio Brogi, Sara Corfini and Stefano Iardella: From OWL-S Description to Petri Nets. E. Di Nitto and M. Ripeanu (Eds.): ICSOC Workshops, pp. 427-438. Springer-Verlag (2009)

[25] Yu, B., Coll, S.: Verifying web services of OWL-S with Petri net. In: 5th IEEE International Conference on Computer Science and Education (ICCSE), pp. 891- 896 (2010)

[26] Vidal, J.C., Lama, M., Bugarín, A. : Towards the use of Petri nets for the formalization of OWL-S choreographies. In Knowledge and Information Systems. Springer-Ed (2011)