ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 1, Issue 9, November 2013

Realtime User Interface Generation in Ubiquitous Environment

Imen Ismail, Faouzi Moussa

CRISTAL Laboratory, National School of Computer Sciences, Manouba University, Tunis

ABSTRACT: In ubiquitous and intelligent environments, the human-system interaction has become a major concern. Consequently, a new form of interaction is emerging, the ubiquitous interaction. Within this context, a system that performs at realtime and as required by the user would support the reliability and reinforce the purpose of such interaction. Adapting the information to the context of use by knowing at the right time the appropriate useful one remains the most crucial challenge to address. Up until today, there is no agreement on an ideal operating strategy for adaptive systems. However, it is recognized that adaptation strategy is a range of policies designed for the system when responding to a specific event. In the light of this, one of our key policy issues is to extract the information needed to create and update a current user model as well as his environment. A dynamic modelling approach has been described in this paper. As such, based on the currently used technology, the dynamic composition of web services, this work demonstrates how a task model will be dynamically generated, thus, providing the basis for the automatic and realtime generation of the user interface.

Keywords: Current Model Construction, Dynamic UI Generation, Web Services, OWL-S, Ontologies

I. Introduction

This paper outlines the use of Web services discovery technologies to build a real-time model of the user-system interaction. The ultimate goal is to deduce from this model the contents of the mobile user interface in a ubiquitous environment. The proposed approach is particularly based on the semantic Web service matching. The match and discovery engine applies this approach to dynamically create the model by discovering and selecting the relevant elementary models which are specified in OWL-S atomic services. A generic algorithm of this engine can be a valuable help in understanding how it works. Specifically, we show how a user's elementary action that meets certain selection criteria can be transformed into a service which will be processed by this engine to select the expected elementary model and thus to build the whole model.

Web services are defined by W3C as "a software system designed to support interoperable machine-to-machine interaction over a network" [1]. Web service discovery plays a key role in finding appropriate Web services. The challenge is to discover accurate Web services in accordance with users' requirements [2]. Semantic Web services promise the combination of semantic Web with Web service technology in order to overcome the limitations of current Web services by adding explicit semantics to them [3][4][5]. In particular, semantic Web Services promise to provide solutions to the challenges posed by the automated discovery, dynamic composition, enactment, and other tasks associated with managing and using service-based approaches [6][7]. Semantic Web services aimed at providing formal descriptions of requests and Web services that can be exploited to automate several activities in the Web services usage process, including dynamic discovery of services. The interoperability of autonomously developed and deployed Web services have greatly benefited from the notion of semantic Web services [8]. Discovery of services is the process of finding Web services Provider locations which satisfy specific requirements.

Our approach doesn't aim at providing a conceptual model for semantic Web service discovery but it exploits this key feature to implement what we refer to as dynamic composition of a realtime model. In this paper, we investigate the implementation of a discovery engine that can provide dynamic model discovery. In effect, this model describes the user-ubiquitous system interaction.

Ubiquitous systems are systems that aim at providing the suitable information relevant to the interaction with mobile users. They will be able to adapt dynamically their behaviours based on the context change of users. This ability consists in perceiving the environment, detecting the contextual parameters and dynamically reacts to the environment

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 1, Issue 9, November 2013

variations to satisfy the realtime user's needs and requirements [9] [10] [11]. In our study, we particularly investigate the context of use generated by the user's activity's change.

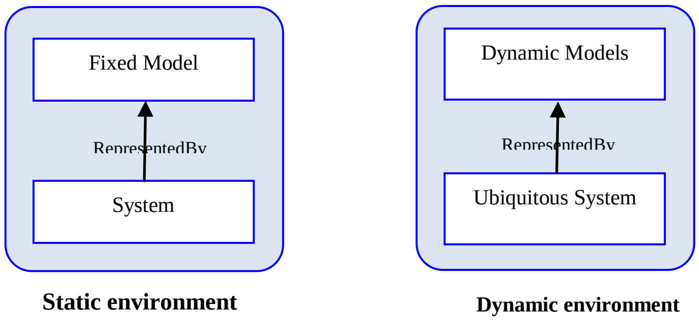


Fig. 1 New requirements for UI-S modelling in highly dynamic environments

Our main retrieval mechanism is to build realtime and dynamic user interaction models (Fig.1). Thus, the system could have a suitable representation of the user-system interaction in a particular situation on realtime. Hence, it can deduce the suitable model or build a realtime one according to the current context of use. It can then adaptively present to the user the relevant information concerning what he is doing. Our method is to divide the model of user activity in sub-basic activities that we call "*Elementary actions*". Then, we represent these elementary actions with owl-s atomic services. Finally, the realtime construction of such dynamic models is none other than the automatically and dynamically discovery and invocation of services.

II. Semantic Web Services and OWL-S Language

Semantic Web Services promise to provide solutions to the challenges associated with automated discovery, dynamic composition, and enactment [12]. OWL-S is a specific owl-based Web service ontology [13]. It supplies a core set of markup language constructs for describing Web service in computer-interpretable form. The OWL-S ontology is designed to provide a framework for semantically describing such services from several perspectives (e.g., discovery, invocation, composition). Each service is characterized by three main concepts: Profile, Process and Grounding (Fig.2).

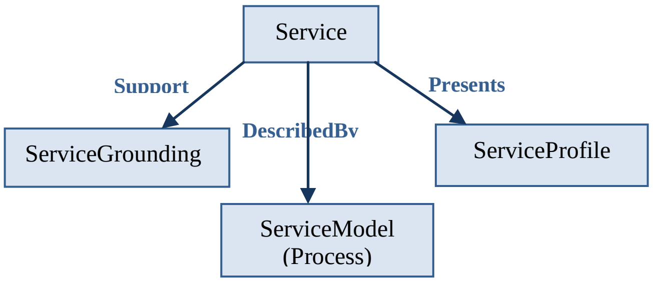


Fig. 2 Representation of the service ontology

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 [14]. The Grounding specifies on how to interoperate with a service via messages [15]. 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. And the Contact Information that provides a mechanism of referring to humans or individuals responsible for the service.

The abstract procedural concepts provided by OWL-S ontology can be used along with the domain specific OWL ontologies [12]. These concepts are characterized as an IOPR model (Inputs, Outputs, Preconditions and Results) which

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 1, Issue 9, November 2013

provide the terms, concepts, and relationships used to describe various service properties (Fig. 3). The Inputs are required data for the service execution, whereas outputs are data 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*; it 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, which 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 [1]. 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.

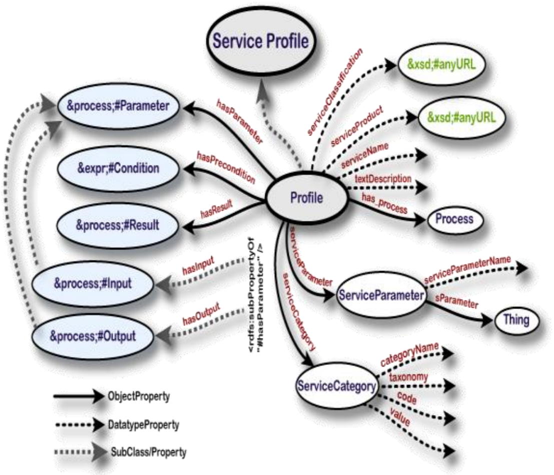


Fig. 3 Parametric structure of OWL-S service profiles

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.

III. CONCEPTUAL AND FUNCTIONAL FRAMEWORK OF THE PROPOSED ADAPTATION STRATEGY

In the interest of getting appropriate services, the system queries the **EARegistry** register. The service discovery is, in fact, achieved through its description in the **EARegistry**. Following acquisition of the required service, the system could easily retrieve the needed information. In other words, based on the service description it may make binding and call service functions, including "*UserRequirements*" Parameter.

Fig. 4 depicts a detailed and analytical view of the various architecture components [16][17]. The emphasis is on "ChainMaker" component especially given where he is the core architecture. It shall ensure the discovery mechanism and it will dynamically build the model's file using semantic specification of concepts defined by the domain ontology.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 1, Issue 9, November 2013

There are three distinct phases prove to be crucial to dynamically create a model's sequence. The next EA necessary to meet the system's need must to be discovered at first stage. Secondly, the most relevant EA must be selected. Finally, perform the matching operation in order to retrieve results from the EA which is being processed. To reach these targets, we proposed to use semantic descriptions of web services, particularly "precondition" and "results" concepts of OWL-S specification. The suggested solution can be described in the steps below.

- Through the system query analysis, identify the required functionality is the first step.
- Secondly, thanks to their semantic description stored in the ***EARegistry***, we must locate the appropriate elementary actions (***EADiscovery***) to be useful for the current task processing.
- Select the (SEA) that best fits the system requirements (***EASelection***) is the afterwards step.
- Then, we must combine the selected services in order to satisfy targeted need (***ChainMaker***).
- All that remains is the final stage, get results given by the SEA (Requirement Deduction).

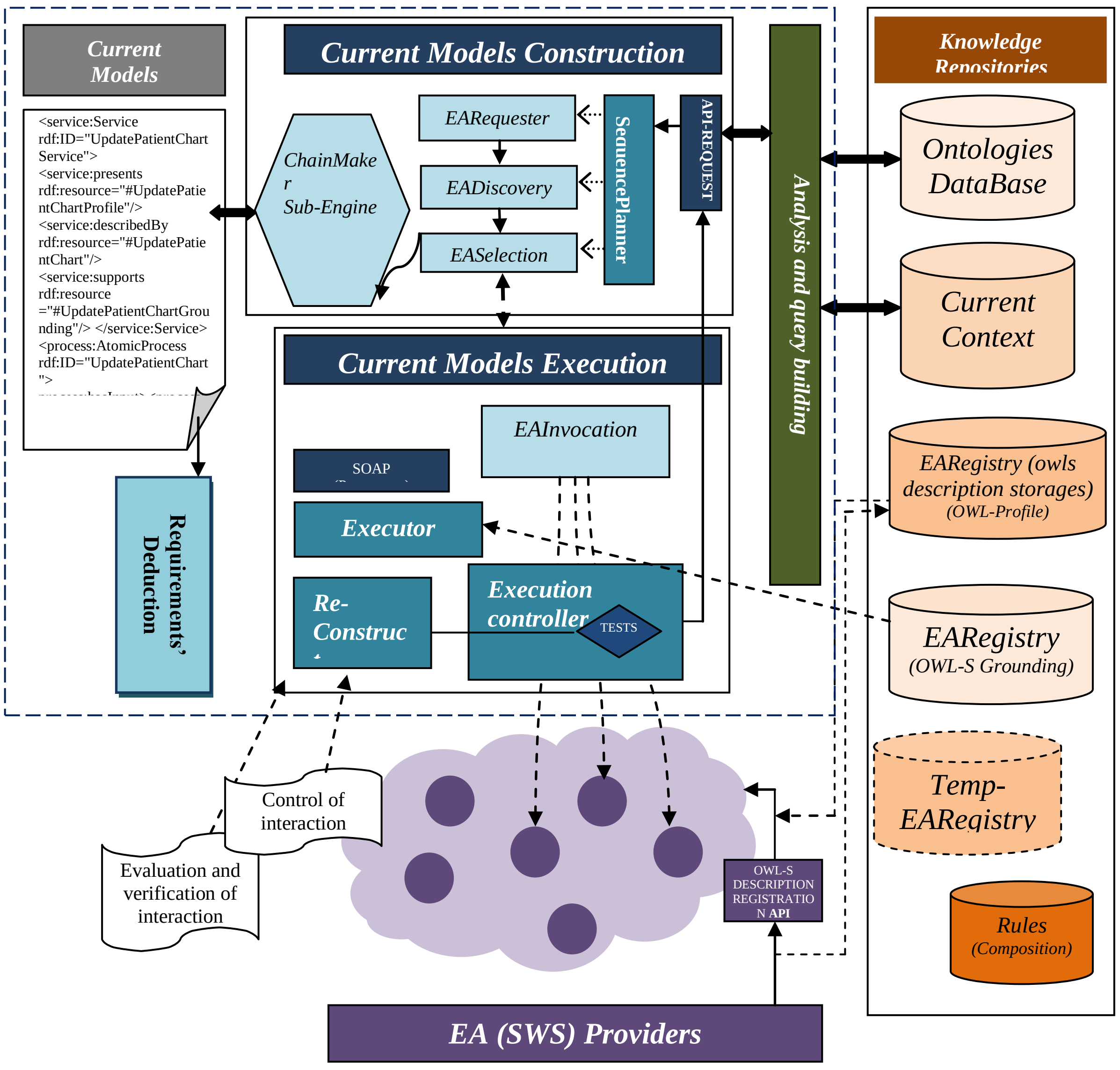

Current Models
<service:Service rdf:ID="UpdatePatientChartService"> <service:presents rdf:resource="#UpdatePatientChartProfile"/> <service:describedBy rdf:resource="#UpdatePatientChart"/> <service:supports rdf:resource ="#UpdatePatientChartGrounding"/> </service:Service> <process:AtomicProcess rdf:ID="UpdatePatientChart">
Requirements' Deduction
Current Models Construction
ChainMaker Sub-Engine
EARequester
EADiscovery
EASelection
SequencePlanner
API-REQUEST
Analysis and query building
Current Models Execution
EAInvocation
SOAP
Executor
Re-Construct
Execution controller
TESTS
Knowledge Repositories
Ontologies DataBase
Current Context
EARegistry (owls description storages) (OWL-Profile)
EARegistry (OWL-S Grounding)
Temp-EARegistry
Rules (Composition)
Control of interaction
Evaluation and verification of interaction
OWL-S DESCRIPTION REGISTRATION API
EA (SWS) Providers

Fig. 4 Adaptation System: reduced functional architecture

The proposed architecture covers, thus, the following key components which work well in harmonic combination: *(i)* a "***chainMaker***" which ensures the current model chaining. *(ii)* A "Knowledge Repository" that stores data and ontologies, inter alia, service semantic descriptions which were presented by a OWL ontology, more particularly "***EARegistry***". *(iii)* The criteria useful to select the "SEA" will be identified and analysed by the "***EARequester***" component. However, the corresponding API "EARequester API" will transform the initial query description into a pseudo-description by referring to terms and concepts of the domain ontology. This way, whenever an action is invoked through its input parameters, it will return the output parameters as results.

IV. ALGORITHMS

In this section, we conduct the analysis and development of algorithms that reflect the adaptation strategy focus on desired objectives. For this purpose, three main operational cases have been considered:
(a). *Normal Execution Mode*: This mode is considered as the default case and the system shall be in normal operating under this mode.
(b). *Re-planning Execution Mode*: It would take place in a context change.
(c). *Planning Execution Mode*: Supposed to be held during unknown case management.

A. Normal mode algorithm

This algorithm (Fig. 5) presumes the presence of normal execution environment. It rightly starts from situation analysis and formulate the corresponding query. The "*SequencePlanner*" module retrieves the "owl-s Process Model" concerning this specific request. Then, it parses the file content and extracts two main lists. The first one includes basic actions to manage the current situation. The sequencing of these actions is organized in the second extracted list.

```
Algorithm PseudoCurrentModel
A.    Find OWL-S Process Model that best mutch with the current situation
B.    Extract two data lists from the file (owl-s)
Returns (List) EAList : set of ordered EA
        (List) SeqList : set of used control structures
    1:    Function PseudoCurrentModel
    2:    Local EAList, SeqList, (String)DataSet, (Int) LineNb=0
    3:    Begin
    4:    (owl-s) CurrentSolution=LocalDiscovery(R) /*a discovery procedure
          (local) call specifications owl-s responding to the concerned request */
    5:    While(Mode=0) /*Mode is a global Var*/
    6:    If notEmpty(CurrentSolution) then /*Parse the file*/
    7:    While (NOF(CurrentSolution))
    8:    DataSet = Line(CurrentSolution, LineNb)
    9:    EAList := EAList ∪ extractEA(DataSet)
   10:    SeqList := SeqList ∪ extractSeq(DataSet)
   11:    End While
   12:    End While
   13:    Return EAList, SeqList
   14:    End PseudoCurrentModel
```

Fig. 5 Normal mode algorithm

B. The other operational algorithms

The second operation mode, i.e. "*ReplanningExecutionMode*", is defined by the below mentioned algorithm. This algorithm will be triggered when whether the context of use changes. In such a case, the initially pre-selected EA is likely to become inappropriate. The corresponding Elementary Actions the most suitable for the new situation management must be re-planed. This new planning is based on discovery and selection approaches. These particular tasks are the responsibility of the "SequencePlanner" module. This mechanism remains the same for the third mode, although this mechanism begins in this way from the operation start.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 1, Issue 9, November 2013

C. EA preparation algorithm

Initially, the basic actions - intended to be used by the adaptation strategy as atomic or composite web services - begin loading in the temporary repository (T:*TempEARegistry*). This pre-selection (Fig. 6) is based on an analysis of contextual information [16] relating to usual activities, profil of the user, etc. These parameters constitute the first request (***InitRequest***). Therefore, both the extraction performance invocation of EA as well as the execution time will be improved.

```
Algorithm (0) EAPreselection
Find EAi that best match with the initial Request (InitRequest) of the adaptation system
Returns set of EA
1:    Function EAPreselection(InitRequest IR)
2:    Local FirstResult ;
3:    Begin
4:    For All (EA) Do
5:    While (EA.hasCategory = IR.Category) OR (EA.hasActivity=IR.Activity)
      OR (EA.hasProfile=IR.Profile)
6:    FirstResult := FirstResult ∪ {EA}
7:    Return FirstResult
8:    End EAPreselection
```

Fig. 6 The Pre-selection algorithm

After the pre-selection step, comes loading step which is done in temporary repository. Then, in the light of what has been selected above, the location and the recovery of candidate EA will progress in most efficient and effective way. Now we will discuss the selection (Fig. 7) of the next EA that fit the request and the ranking step using a pre-extracted set of criteria, of which requirements are developed in the subsequent algorithms.

```
Algorithm (1) EASelection
Find EAi that best match with the Request (SysRequest) of the adaptation system
Returns set of EA (Ordered)
Notation : EAc = {EA ⊂ T, avec la même Catégorie}
1:     Function EASelection(SysRequest SR)
2:     Local CurrentResult ;
       Local PriorityDegree ;
3:     Begin
       /*the algorithm begins by extracting any feature of priority on which
       he will order the EA*/
4:     PriorityDegreeSys := extractPD(SR)
5:     For All (EA ∈ EAc) Do
6:     If ConceptSim(SR, EAi)
         Then
7:       CurrentResult := CurrentResult ∪
         InsertOrdered(CurrentResult, EAi)
8:     End if
9:     End For
10:    Return CurrentResult
11:    End EASelection
```

Fig. 7 The EA Selection algorithm

The second step of selection process is based on semantic comparison of concepts, while considering that Inputs and Outputs refer certainly to domain ontology concepts. This phase is based on semantic similarity between the query and the EA descriptions. In this respect, we found it particularly helpful in referring to the approach proposed by [18] and approved by [20] method [19] to calculate the semantic similarity of two concepts X and Y: ConceptSim (X, Y).

- ConceptSim (Place, PaCarbohydrateIngest) =1 (étant donné que PaCarbohydrateIngest est une instance de la classe Place),
- ConceptSim (Transition, BeginCarbohydrateIngest) =1 (étant donné que BeginCarbohydrateIngest est une instance de la classe Transition).

D. Model composition and execution algorithms

The below-described algorithm (Fig. 8 a&b) allows composite service generation from the description given by *EASelection* operation. The algorithm is based on composability rules key to understanding how two EA are composable (e.g. Sequential, Parallel, Choice, etc.). Two composability rule types where identified: (i) Syntax rules which include rules of operation's types and protocol connections between EAs (i.e. the bindings). And (ii) semantic rules including rules that control exchanged message compatibility and service semantic domain compatibility. Technically speaking, however, the algorithm has direct impact on the current execution model file (CurrentModel.owl). Having the current EA as input in one hand, and how to add this EA to the current model on the other hand, the algorithm allow the EA integration in the whole current model while checking some properties.

```
Algorithm Chainmaker
It tries to update the model running by adding, deleting, or modifying an elementary
action
Returns the current model in execution "CurrentModel.owl"
1:    Function Chainmaker (CurrentEA,MMode)
2:    Begin
3:    While (MutEvent := False)
4:    /* Open the owl file corresponding to the current elementary
      action*/
   owlDoc = Load ("EA.owl ") ;
5:    /* Merge two files => add selected content of the current action
      at the end of file CurrentModel*/
      Merge(CurrentModel, CurrentEA, MMode) ▷ Function Call
6:    FileStorage(CurrentModel) ▷ Function Call
7:    Out(CurrentModel) /*File Out*/
8:    End Function
```

Fig. 8 (a) Current Model Construction algorithm

```
Algorithm  Merge
It tries to Merge two files
Returns  Updated file
1:     Function  Merge(CurrentModel, CurrentEA, MMode)
2:     Local ContentToUpdate= ""<process:composedOf>" (String)
3:     Begin
4:   ReachEOF(CurrentModel) ;
   /*Reading file until reaching the end of the file :
   ("</process:composedOf></process:CompositeProcess>")*/
5:     a return line and add the tag according to MMode
   Switch (MMode)
6:      Begin
7:   Case of  "Choice"
8:   ContentToUpdate = ContentToUpadte + "<process:Choice>"
9:   Case of  "Seq"
10:  ContentToUpdate = ContentToUpadte + "<process:Sequence>"
11:  Case of  "Par"
12:  ContentToUpdate = ContentToUpadte + "<process:Parallel>"
13:    ...
14:    End Switch
15:    ContentToUpadte= ContentToUpadte+"</process:composedOf>"
16:    End Function
```

Fig. 8 (b) The Merge algorithm (Function)

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 1, Issue 9, November 2013

E. User Interface Generation

A GUI is generally specified by one or more widget(s). A widget was originally defined as a graphical user interface component. Within the ambit of our approach, these objects are dynamically identified and assembled on the fly under the adaptation engine monitoring. In this step, the realtime reorganization as well as the arrangement of necessary and available widget are supervised. Ultimately, our aim is to build the most appropriate and useful GUI (Fig. 9).

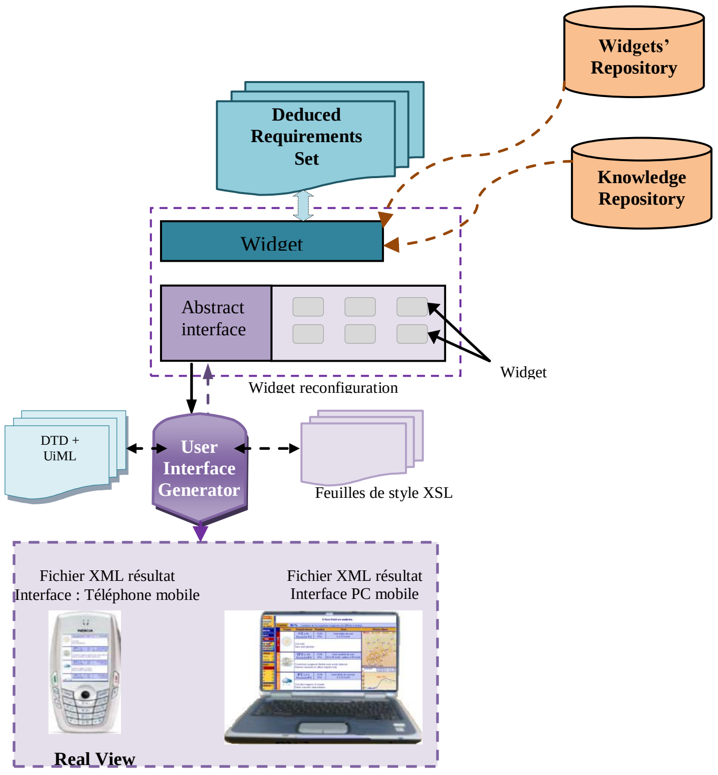


Fig. 9 User Interface Generation Section

1) Content categorization

Our proposed categorization aims at highlighting the type of information according to the user's requirements. The contextual data that help the user by giving him useful information is called Informational Parameters (***InfoParam***). Moreover, the user may need some judicious information to help him make a decision in a particular situation or under collaborative activities. This kind of information is called Decisional Parameters (***DeciParam***). Some information may not be interesting for the user, while it should be heavyweight informative. This kind of information is called Alert Parameters (***AlerParam***). Each of these types will be transmitted to the user interface through different forms according to their specific content as described below:

- Alergets: (Alerting-widgets) the content of these widgets is considered warnings to evoke an emergency state and thus request a direct user intervention.
- Infogets: (Informational-widgets) the content of these widgets is closely related to the user's requirements.
- Decigets: (Decisional-widgets) the content of these widgets consists of judicious data that enable smarter decisions.

Within this classification, the user interface content and presentation can give advice by smarter decisions and calls the user's attention by alerting him about a specific situation. This also may lead to improve the preciseness and the conciseness of the provided information. Finally, based on the Petri-nets model of the user activity and taking into

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 1, Issue 9, November 2013

account the identified contextual parameters, it will be easily to deduce all widgets needed for a useful user interface. This is simply realized by associating the suitable widgets to each corresponding parameter (Fig. 10). Based on the previously described classification, the precision and conciseness of the provided information become much more significant.

2) Content generation

Based on the current user activity model [21], and while taking into account the identified contextual parameters; it would, therefore, be straightforward to derive all widget necessary for a useful user interface generation. This is achieved while matching the appropriate widget with the corresponding parameter.

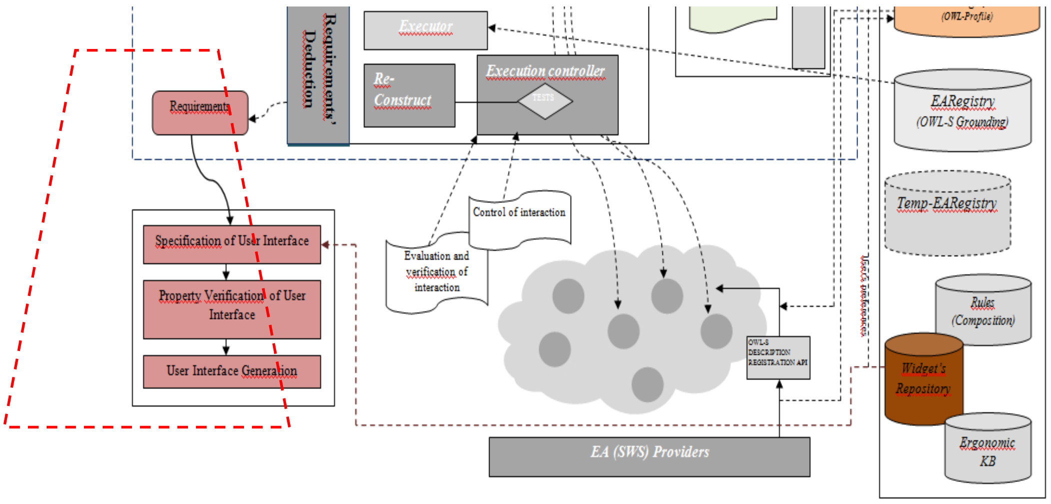


Fig. 10 User Interface Generation Section

It is noteworthy that a third platform implements the interface generator. After retrieving data from the user's device, "User Interface Generation" module should generate the interface (Fig.9 & Fig.10). Then, it should transfer it to the target platform to adapt. This method allows creating interfaces in accordance with the target's capacities and user's preferences and habits. A specific DTD document and XSL Stylesheet will be later used to generate ad-hoc interface. The DTD document is indeed a document that describes an XML model. It defines the allowed tags, their orders and elements sets they define. Thus, only the useful tags will be displayed when generating the interface. Finally, the XSL stylesheet describes how the same DTD XML based documents will be submitted and displayed. The interface model is an interface representation aspect with a certain abstraction degree. Graphical interface information, notation and high-level description are used to reflect the various aspects of this model. However, the final interface is typically composed of source code and all details useful for running on a specific platform. In other hand, a concrete interface merely constitutes a description form that contains presentation details as well as components' layout. With regard to the adaptation, it can be performed when generation the final interface or at runtime.

V. Results

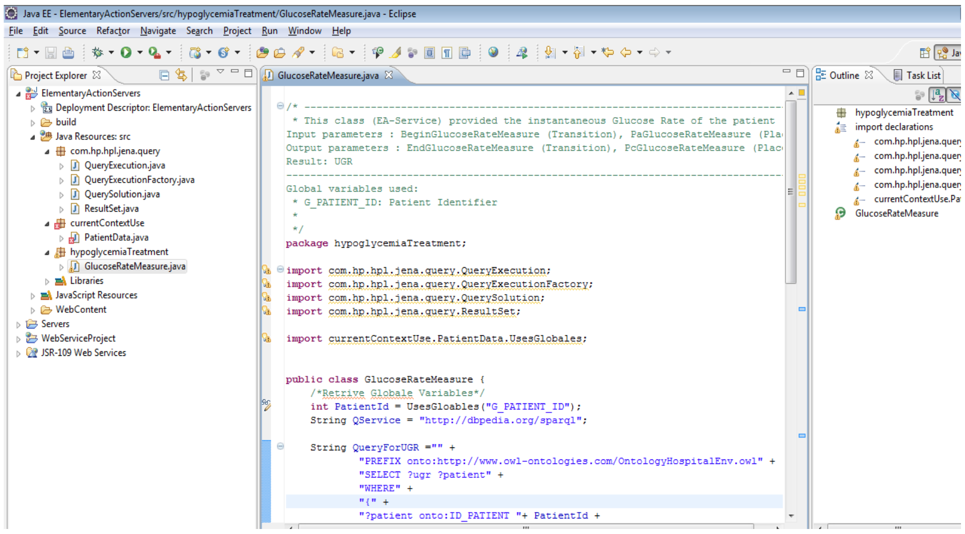
A simple conceptual proposal test is presented. The implementation of knowledge and data used by the whole system was created through the Protégé tool, and will be detailed in future paper. We just present results of test corresponding to user interface generation. To implement the user's task as web service we must first create the operation class that implement this task (eg. Fig. 11). Then we will have to deduce the corresponding WSDL description (eg. Fig. 12).

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 1, Issue 9, November 2013



```
/* ---------------------------------------------------------------------------
 * This class (EA-Service) provided the instantaneous Glucose Rate of the patient
Input parameters : BeginGlucoseRateMeasure (Transition), PaGlucoseRateMeasure (Pla
Output parameters : EndGlucoseRateMeasure (Transition), PcGlucoseRateMeasure (Plac
Result: UGR
---------------------------------------------------------------------------
Global variables used:
 * G_PATIENT_ID: Patient Identifier
 *
 */
package hypoglycemiaTreatment;

import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.query.ResultSet;

import currentContextUse.PatientData.UsesGlobales;


public class GlucoseRateMeasure {
    /*Retrive Globale Variables*/
    int PatientId = UsesGloables("G_PATIENT_ID");
    String QService = "http://dbpedia.org/sparql";

    String QueryForUGR ="" +
           "PREFIX onto:http://www.owl-ontologies.com/OntologyHospitalEnv.owl" +
           "SELECT ?ugr ?patient" +
           "WHERE" +
           "{" +
           "?patient onto:ID_PATIENT "+ PatientId +
```

Fig. 11 Functional part of the Elementary Action

Develop and compile the owl-s description model is the next step. The final step requires (a) the compilation of the process model specific to the current execution of the web service; (b) the compilation of the profile that facilitates discovery. Finally, (c) compile the “Grounding” facet related the process model. Initial evolution tests have shown satisfactory results that best meet our expectation. Indeed, based on the ontology and semantic data, the system is able to deduct at realtime the required information. This required information is the expected outcome of the web service that encapsulates the current task. It’s important to note that only interesting services will be invoked by the system to construct the task model. The deduction of the required information needed to be display (Fig. 13), while the task is being executed, is the retrieval results returned by the WSDL file scanning. The user interface should contain an informational widget (Infogets) that display on the target device’s screen the useful informational data. For this reason, we actually speak about the user interface generation from the web service’s WSDL description.

VI. CONCLUSIONS

The ultimate purpose of the ubiquitous computing is to make the task of the user easy in a very pleasant and useful interface. In particular, it aims at replace his intervention in any time-wasting task. The capacity to build user’s interfaces that support these purposes in ubiquitous environment has attracted quite a bit of attention. In this paper, we have presented an approach suitable for continually updating the user’s changing requirements, thus ensure a realtime adaptation of the provided information to suit needs, preferences and distinct behaviours of different users. The objective is to make and apply on-the-fly deductions and decisions about which widgets will be displayed on the mobile devices. With the use of owl-s technology, the required information – is actually supplied by web services – can be made more efficient and more dynamic. With a simple test example, we have focused on how to deduce the required information from the elementary task WSDL description; therefore how to generate user interface content.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://hypoglycemiaTreatment"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://hypoglycemiaTreatment"
xmlns:intf="http://hypoglycemiaTreatment"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
```

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 1, Issue 9, November 2013

```
<wsdl:types>
  <schema elementFormDefault="qualified"
targetNamespace="http://hypoglycemiaTreatment"
xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="returnUGR">
    <complexType><sequence>
      <element name="T1" type="Transition"/>
      <element name="P2" type="Place"/>
     </sequence></complexType>
   </element>
   <element name="returnUGRResponse">
    <complexType><sequence>
      <element name="returnUGRReturn" type="xsd:float"/>
     </sequence></complexType>
   </element>
  </schema>
 </wsdl:types>
   <wsdl:message name="returnUGRResponse">
      <wsdl:part element="impl:returnUGRResponse" name="parameters"/>
   </wsdl:message>
   <wsdl:message name="returnUGRRequest">
      <wsdl:part element="impl:returnUGR" name="parameters"/>
   </wsdl:message>
   <wsdl:portType name="GlucoseRateMeasure">
      <wsdl:operation name="returnUGR">
         <wsdl:input message="impl:returnUGRRequest" name="returnUGRRequest"/>
         <wsdl:output message="impl:returnUGRResponse" name="returnUGRResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="GlucoseRateMeasureSoapBinding"
type="impl:GlucoseRateMeasure"><wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="returnUGR">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="returnUGRRequest">
         <wsdlsoap:body use="literal"/>
         </wsdl:input><wsdl:output name="returnUGRResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="GlucoseRateMeasureService">
      <wsdl:port binding="impl:GlucoseRateMeasureSoapBinding"
name="GlucoseRateMeasure">
         <wsdlsoap:address
location="http://localhost:8080/ElementaryActionServers/services/GlucoseRateMeasur
e"/> </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Fig. 12 WSDL generation of elementary action

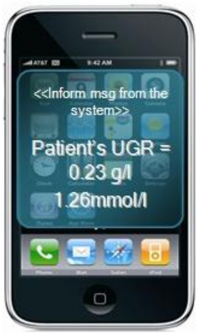


Fig. 13 User interface generation example with the required information

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 1, Issue 9, November 2013

REFERENCES

1. Owl-s Semantic Markup for Web Services, available at: http://www.w3.org/Submission/OWL-S/. Last update 22 November 2004. Last consultation October 2012.
2. Hou, Jun, Zhang, Jinglan, Nayak, Richi, & Bose, Aishwarya, "Semantics-based Web service discovery using information retrieval techniques". In Comparative Evaluation of Focused Retrieval. Springer-Verlag Berlin Heidelberg, pp. 336-346, 2011.
3. Ruben Lara, Miguel Angel Corella, Pablo Castells, "A flexible model for service discovery on the Web, Semantic Web Services with WSMO", Special Issue on Semantic Matchmaking and Resource Retrieval, International Journal of Electronic Commerce, Volume 12, Number 2, Pages 11-41, 2008.
4. Matthias Klusch, Patrick Kapahnke, "Adaptive signature-based semantic selection of services with OWLS-MX3", Multiagent and Grid Systems 8(1): 69-82, 2012.
5. Matthias Klusch, Patrick Kapahnke, "The iSeM matchmaker: A flexible approach for adaptive hybrid semantic service selection", J. Web Sem. 15: 1-14, 2012.
6. Uwe Keller, Ruben Lara, Holger Lausen, Dieter Fensel, "Semantic Web Service Discovery in the WSMO Framework", in Semantic Web Services: Theory, Tools and Applications, Chapter XII pp, 281-324, 2007.
7. 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.
8. S. McIlraith, T. Son, and H. Zeng, "Semantic Web Services", in IEEE Intelligent Systems, 16(2):46–53, 2001.
9. 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.
10. Mark Weiser, "some computer science issues in ubiquitous computing". Commun. ACM. 36(7): 74-84, 1993.
11. John Krumm, "Ubiquitous Computing Fundamentals", Redmond, Washington, U.S.A. by Taylor and Francis Group, ISBN 978-1-4200-9360-5, 2010
12. Jyotishman Pathak, Neeraj Koul, Doina Caragea, and Vasant G. Honavar. "A framework for semantic Web services discovery", In Proceedings of the 7th annual ACM international workshop on Web information and data management. ACM, New York, NY, USA, 45-50, 2005.
13. 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, http://www.w3.org/TR/2009/RECowl2-syntax-20091027. Latest version available at http://www.w3.org/TR/owl2-syntax/. Last consultation March 2012
14. Quang, P., Tien, "T.: Ontologies et Web Services". Activity Report. Institut de la Francophonie pour l'Informatique (2005)
15. 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
16. I. Ismail, 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
17. Ismail I. and Moussa F., "Towards a Runtime Evolutionary Models of User Interface Adaptation in a Ubiquitous Environment". The Fifth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies: UBICOMM, Lisbon, Portugal, 2011.
18. N. Ould ahmed, S. Tata, "How to consider requester's preferences to enhance web service discovery", in Proceedings of the second International Conference on Internet and Web Applications and Services, Morne, Mauritius, pp 59-64, 2007.
19. Rohallah Benaboud, Ramdane Maamri and Zaidi Sahnoun, "Agents And OWL-S Based Semantic Web Service Discovery With User Preference Support". International Journal of Web & Semantic Technology (IJWesT) Vol.4, No.2, 2013.
20. Rohallah Benaboud, Ramdane Maamri, and Zaïdi Sahnoun, "Semantic Web Service Discovery Based on Agents and Ontologies". International Journal of Innovation, Management and Technology, Vol. 3, No. 4, 2012.
21. Imen Ismail, Faouzi Moussa, "Dynamic and Realtime Modelling of Ubiquitous Interaction". In The International Conference on Control, Modelling, Computing and Applications (CMCA-2012).