International Journal of Smart Home
Vol.7, No.6 (2013), pp.377-392
http://dx.doi.org/10.14257/ijsh.2013.7.6.37

A Framework and its Associated Process-Oriented Domain Specific Language for Managing Smart Residential Environments

Abdaladhem Albreshne[1], Ayoub Ait Lahcen[2] and Jacques Pasquier[3]

[1, 3]*Computer Science Department, Fribourg University, Switzerland*
[2]*ENSA (Ecole Nationale des Sciences Appliquées), Kenitra, Morocco*
{abdaladhem.albreshne, ayoub.aitlahcen, jacques.pasquier}@unifr.ch

Abstract

Nowadays, researches on Smart Environments represent interesting challenges in pervasive computing. Indeed, remote services, wireless networks and miniaturized devices are becoming more and more diverse and affordable, as well as essential in our daily activities. Unfortunately, the most of these handled devices and services are rather isolated and cannot easily cooperate between each other. Moreover, there is a clear lack of well accepted standards and platforms to help integrate them into useful process-oriented composite applications. As a consequence, services composition and orchestration research is exploring ways to combine available services and smart components to work together for solving complex problems such as energy control, security, health care, etc. The present paper aims to explore the previous challenges in the context of controlling Smart Residential Environments (SRE). The proposed solution consists first in defining a generic ontology of such environments and then in using it within a Domain Specific Language (DSL) for designing and controlling different environment processes.

Keywords: *Smart Residential Environments, Ontology, Semantic Web Services Composition, Domain Specific Language, BPEL4WS, OWL-S, WSDL*

1. Introduction

Nowadays, researches on Smart Environments represent interesting challenges in pervasive computing [1]. To deal with these challenges, many development paradigms have been suggested in the literature [2]. One of them is Services Oriented Computing [3], which has been proposed as a solution for heterogeneous and changing environments. In spite of that, a number of challenges in controlling Smart Environments are still being explored by both industrial and academic communities. We distinguish two main problems in this context. The first one is how to describe the environment and devices that users, respectively, live in and interact with. The second one is the problem of modeling, composing, orchestrating and executing services in SRE, allowing users to control it. The present paper aims to provide solutions to these problems and proposes a generic global software framework for managing SRE. This framework embodies different elements that have been developed as follow.

Firstly, we have increased the accessibility of physical entities in SRE by providing well-defined programming interfaces. These interfaces include semantic support to give a high level common understanding of the environment. Secondly, we have implemented a service discovery engine in order to find automatically, in a SRE, the intended services and their providers. Thirdly, we have developed an original Domain Specific Language for designing and controlling the environment processes. It also allows us to create ready-to-use templates for specific goals (*e.g.*, energy saving). Finally, we have implemented a process execution engine for our DSL.

ISSN: 1975-4094 IJSH

The paper is organized as follows. Section 2 proposes a generic terminology for describing SRE. It will serve as the basis for the ontology and the domain language proposed later in Sections 5 and 6, respectively. These two sections constitute the core of the paper and they are introduced by a motivating scenario in Section 3 and a discussion about related work in Section 4.

2. Smart Residential Environment: Concepts and Definitions

An entity is a real world object, location or person. We say that an entity is smart when we associate a set of sensors and/or actuators to it. We say that a residential environment is smart when it embodies smart entities. More formally, we define an entity, a smart entity and a residential environment as follow:

Definition 1 (Entity): An Entity is a 2-tuple $E = <c, P>$, where:

- c is the category of the Entity;
- P is a set of parameters.

The category is an object, a location or a person. P has required and optional parameters. The required ones are {*entityID, entityName*}. The optional parameters depend on the category. At a given point in time, the parameters values give information about the entity. We say that they represent the ***entity state***.

Definition 2 (Smart Entity): A Smart Entity is an entity with sensors and/or actuators. It is a 4-tuple $SE = < c, P, S, A>$, where:

- c is the category;
- P is a set of parameters;
- S is a set of sensors;
- A is a set of actuators.

Definition 2.1 (Sensor): A sensor makes available smart entity parameters at a given point in time. It might have associated publishers. It is a 3-tuple $S = <P, Q, U>$, where:

- P is a set of parameters;
- Q is a set of queries;
- U is a set of publishers.

P has required and optional parameters. The required ones are: {*sensorID, sensorName*}. Queries are services provided by sensors. In order to execute a query, a sensor gets the parameters' values of the associated Smart Entity.

Definition 2.1.1 (Publisher): A publisher produces a set of events.

Definition 2.1.1.1 (Event): An event occurs when a parameter value of a Smart Entity changes. An event is detected by a sensor and published by a publisher associated to this sensor.

Definition 2.2 (Actuator): An actuator allows modifying smart entity parameters at a given point in time. It is a 2-tuple $A = <P, C>$, where:

- P is a set of parameters;
- C is a set of actions.

P has required and optional parameters. The required ones are: *{actuatorID, actuatorName}*. Actions are services provided by actuators. In order to execute an action, an actuator modifies the parameters' values of the associated Smart Entity.

Definition 3 (Smart Residential Environment): A Smart Residential Environment SRE is a set of Smart Entities.

3. Motivating Scenario

We explore the challenges of controlling SRE through a case study realized in the context of a smart home environment. It will serve as a guiding thread for the remaining of this paper. In this case study, we consider a smart home (see Figure 1) populated with a set of smart entities, including:

1. *Smart Room*. Each room (A, B, C) has two sensors. The first one is a Presence Detection Sensor, which detects the user presence and publishes an event when the associated state changes. The second one is a Temperature Sensor, which monitors the temperature and publishes an event when the associated state changes.
2. *Smart Entrance Door*. The home entrance door has an Access Controller Actuator, which provides the actions *open* and *close*. The door has also an Access Sensor to detect if somebody enters or leaves the house.
3. *Smart Lamp*. Each lamp in rooms (A, B, and C) has a Controller Actuator to switch it on or off.
4. *Smart Heater*. Each heater in rooms (A, B, and C) has a Controller Actuator to switch it on or off, and adjust the target temperature.
5. *Smart Shower*. The shower has a Controller Actuator to open and close the shower faucet. It has also a Faucet Sensor to detect if it is open or closed, and to publish an event when the associated state changes.

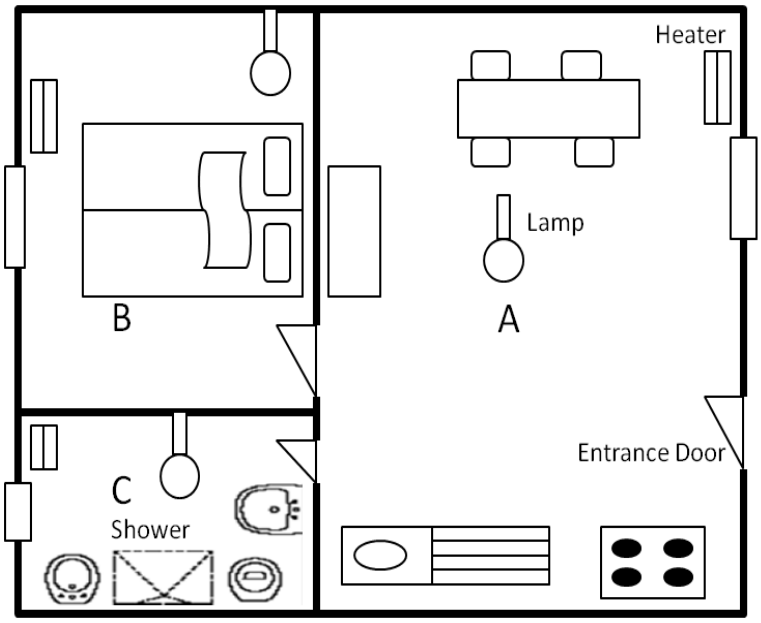

Figure 1. The Smart Home of our Scenario

Table 1 gives an example of how a Home Control System could use these smart entities in order to save energy (other possible examples would be: comfort, surveillance and access control, security, healthcare, *etc.*).

Table 1. Energy Saving Scenario

Time	Description
T1	The faucet sensor informs the Home Control System that Jane starts her shower in Room C.
T2	The Home Control System triggers a timer to wait for 10 minutes.
T3	After 10 minutes, if the shower is still open, the system forces closing it.
T4	The Presence Detection Sensor informs the Home Control System that Jane quitted Room B.
T5	The Home Control System waits for 3 minutes, then switch off the lamp of Room B.
T6	The Temperature Sensor informs the Home Control System that there is a significant temperature change in Room A.
T7	The Home Control System checks if the current temperature in Room A is more than 18 and less than 22. Otherwise, the system takes the appropriate actions to bring the temperature between this interval.
T8	The Access Sensor of the entrance door informs the Home Control System that Jane quitted the home.
T9	The Home Control System configures the house in the best way to save energy taking into account Jane stored preferences and that there is nobody at home.

3.1. Services Orchestration Requirements and Challenges

Even if the previous scenario is quite simple, it demonstrates that there are some requirements to be satisfied in a home energy saving process. We present hereafter examples of such requirements:

- There is a need to define a generic model of the smart space in order to clearly answer questions such as: what are the actuators and the sensors present in a home; to which smart entities are they attached to; and which are the services (actions, queries and/or events) that they might provide?
- The functionalities of the actuators and sensors are implemented in different ways depending on the constructors. For example, to switch on a lamp, a constructor might provide an *on* operation, while another would call it *switchOn*. The question now being asked is how to enable a standard access to all those different physical components of a Smart Environment. A response to this question can be a middleware layer providing a service abstraction and application programming interface to these physical objects.
- Since the system is designed to control services which are not known at the design time, hardware components (actuators and sensors) need a mechanism to describe themselves and their services. Thus, there is a need for an ontology-based [4] framework, which will help services' discovery and composition by providing them a standardized semantic and identification.
- The decision making process could be a conditional set of actions that need to be taken in order to achieve the desired goal. For example, for a scenario where an inhabitant leaves the home, the decision maker could determine that it should turn off all the lights and reduce the temperature of the heating systems. Unfortunately, there is a lack of process languages which enable involved services, provided by actuators and sensors, to be under the control of a central process.
- Process languages should provide control flow statements (while, if, pick, flow, *etc.*,), which allow for creating complex orchestration control scenarios.

- Processes should support event handlers. Event handlers allow processes to take into account events produced externally.
- Finally, the end-user should be able to interact with the system to reconfigure and customize the processes according to his needs.

4. Related Work

In the last years, several works have been proposed to address, from different points of view, the above presented problems. For instance, to ensure low-level communication, industrial approaches have proposed a number of physical and transport network protocols like Zigbee [5]. They have also provided middlewares that are deployed over those protocols to enable controlling devices' activities and data sharing [6, 7]. Such middlewares provide a service abstraction and application programming interface to facilitate and standardize the accessibility among the different environment devices. However, from the Smart Environment service discovery point of view, hardware components still need more mechanisms to describe themselves and their services in order to be automatically discovered and used.

At the application development level, a number of attempts related to the control of Smart Environments have been made. Manual composition approaches are generally used in a situation where the user has a well-defined process model [8] [9]. Processes are defined using a process execution language like BPEL4WS [10] or BPMN [11]. The problem with this approach is that it forces users to make tedious efforts in finding and understanding detailed knowledge about a lot of web service resources. Automatic composition approaches (without human involvement) are used when the user has no process model, but has a set of constraints and preferences. Several of such approaches have been introduced in the literature [2], *e.g.*, those based on Linear-time Temporal Logic (LTL) [12], Artificial Intelligence planning [13], or Rule-Based planning [14]. However, automatic composition is still viewed as a task of high complexity because of the rapid proliferation of available services to choose from, and the risks of drifting away from the initial user's goal. The third approach (which is close to ours) is called semi-automatic or interactive composition [15, 17]. In this kind of composition, the system usually helps users to find, filter, and integrate automatically the desired services by matching the users' requests with the available services. In addition, it enables end-users to interact continuously during the composition process. Some efforts like [18, 19] use semantic description and ontology to help in improving both discovery and composition operations. For further reading, [2] and [20] give a comparative review of services discovery, semantic annotation and arbitrary service composition techniques for Ambient Intelligence environments.

Finally, from the design point of view, applying software engineering concepts on Smart Environments is still in its early stage. Some initiatives have been taken such as activity diagram [21] and state machines [22]. Our approach is highly promising in this regard. We propose a modular, generic and extensible software architecture (see Figure 2), which takes into consideration semantic aspects of a given Smart Residential Environment in order to build services composition scenarios to control it.

5. The Global Software Framework Architecture

We propose a global software framework solution to discover and use more efficiently the services provided by a given smart environment. Figure 2 gives an overview of our framework. It embodies the following elements and functionalities:

- The user can choose a recommended generic process template from the *Process Templates Repository*. Templates are defined using pluggable domain ontologies, as well as our own domain specific language, which will be presented in Section 6.
- The *Process Generator* converts the chosen template into an executable process.
- At the execution stage, the concrete services of the given SRE have to be discovered according to their ontology.
- It is possible for the final user to precise his context preferences through a simple *Client User Interface*.
- The *Execution Engine* (*e.g.*, Oracle BPEL Process Manager [23]) is the runtime environment.

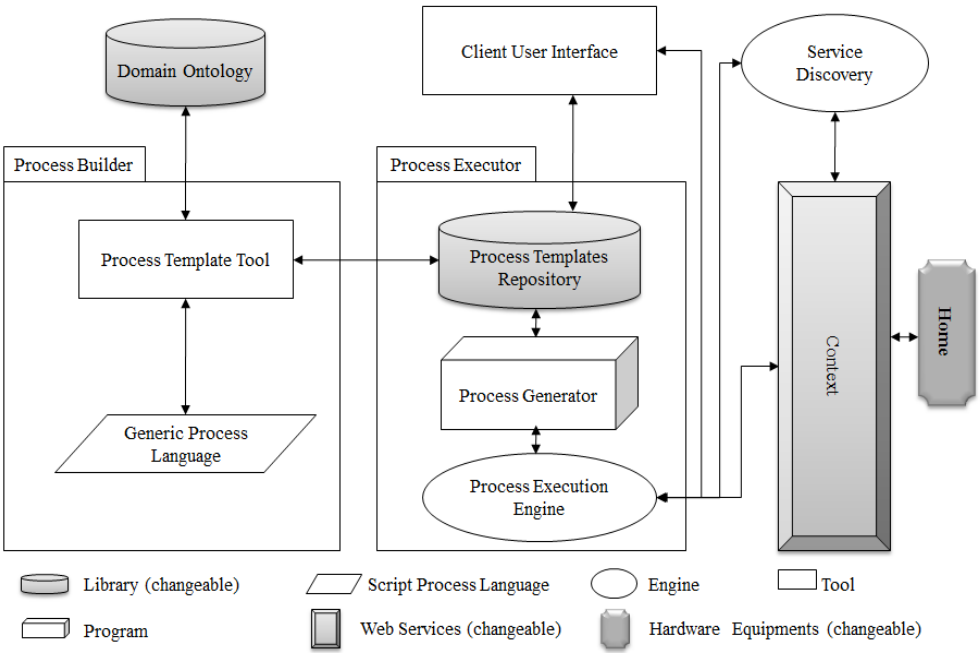

Figure 2. The Proposed Software Framework for Smart Residential Environments

There are two strong (but "reasonable") assumptions in our approach:

- There is an ontology for the environment we want to control. It is needed to have a clear semantic description of the entities' properties and capabilities. It should describe the home resources, including household appliances, lights, windows, doors, etc., as well as their functions, such as turn-on/off, get-temperature, etc. Using such an ontology allows for discovering the needed services and for composing them correctly.
- Each service of a device (either sensor or actuator) is provided through a WSDL [24] interface and proposes an OWL-S [25] description to discover and understand its operations in the context of the domain ontology.

5.1. Ontology for Discovery and Orchestration

An ontology is used to share a common view of a smart space. It defines how actuators and sensors are related to smart entities (*location*, *person* or *object*) and which services (*action*, *query* or *event*) they provide. Using a common ontology ensures a good understanding of services and provides groundings to access them. According to the terminology of Section 2, our SRE ontology is modeled using the

following concepts: *Entity*, *Actuator*, *Sensor*, *Publisher* and services: *action*, *event*, and *query* (see Figure 3). These ontology concepts have the same definitions as those of Section 2.

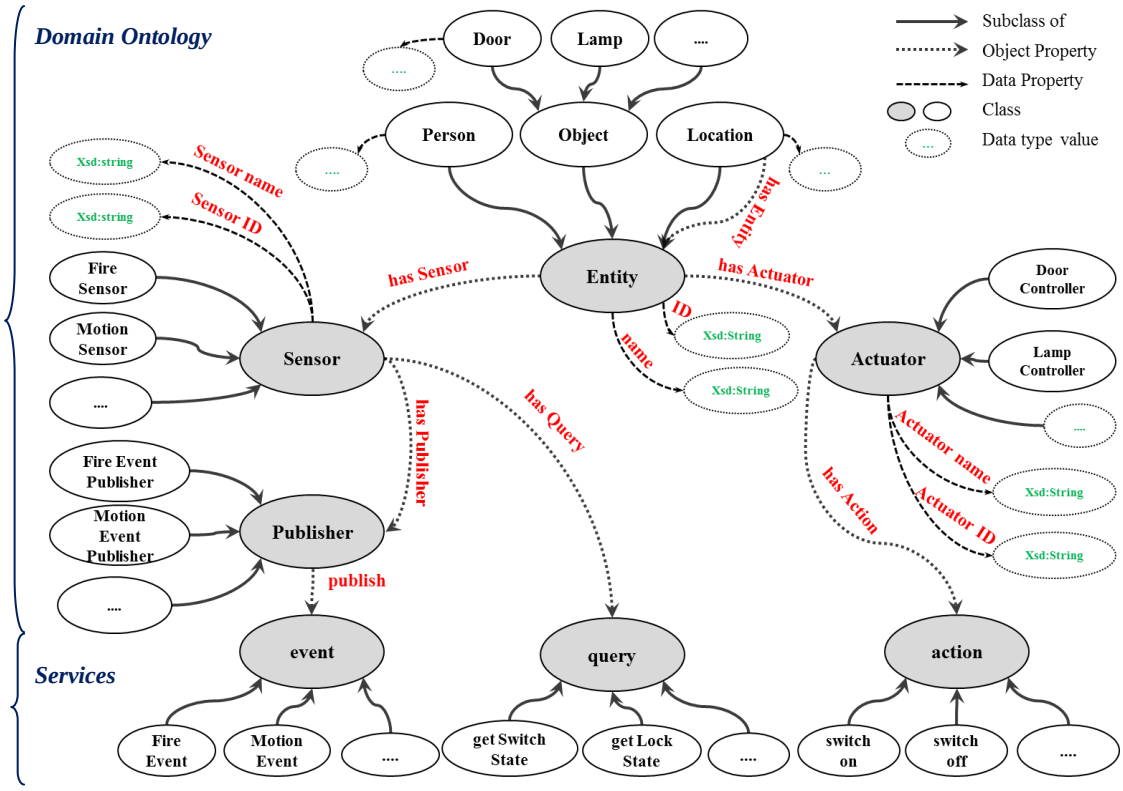

Figure 3. Smart Residential Environment Ontology

We use OWL [4] (Web Ontology Language) to represent our SRE ontology. Each concept of the ontology is represented as an OWL Class. The hierarchy of Entities is clearly defined. Subclasses of the Entity concept are Location, Object, and Person. Subclasses can be further extended. For example, Lamp is a subclass of Object. The hierarchies of Actuators, Sensors, Publishers and Services (*i.e.*, action, event and query) Classes are defined similarly. For example, switchOn and switchOff are subclasses of the action class. We have introduced additional property elements in order to enrich the model: An object Property links a class instance with another class instance, while a Data Property links a class instance with data values. For example, the Object property hasAction links an Actuator instance with a specific Action instance (*e.g.*, a *LampController* instance with a *switchOn* instance). This enables the inference engine to automatically construct a representation of the real smart home, including its smart entities with their actuators and sensors.

As an example, an extract of our SRE ontology, showing the relationship between a smart entity *Lamp*, an Actuator *LampController* and the corresponding services (*switchOn* and *switchOff*), is given in Figure 4.

- Lines 1-13 define a *Lamp* as a type of the class *Electric*, which is itself a subclass of the *Entity* class. A *Lamp* might have a *LampController* actuator.
- Lines 14-31 declare that a *LampController* could have *switchOn* and *SwitchOff action (*without parameters).

```
1   <owl:Class rdf:ID="Lamp">
```

```
2    <rdfs:subClassOf rdf:resource="#Electric"/>
3    <rdfs:subClassOf>
4      <owl:Restriction>
5        <owl:allValuesFrom>
6          <owl:Class rdf:ID="LampController"/>
7        </owl:allValuesFrom>
8        <owl:onProperty>
9          <owl:ObjectProperty rdf:about="#hasActuator"/>
10         </owl:onProperty>
11     </owl:Restriction>
12   </rdfs:subClassOf>
13 </owl:Class>
14 <owl:Class rdf:about="#LampController">
15   <rdfs:subClassOf rdf:resource="#Control"/>
16   <rdfs:subClassOf>
17      <owl:Restriction>
18        <owl:onProperty>
19          <owl:ObjectProperty rdf:about="#hasAction"/>
20        </owl:onProperty>
21        <owl:allValuesFrom>
22          <owl:Class>
23            <owl:unionOf rdf:parseType="Collection">
24              <owl:Class rdf:about="#switchOff"/>
25              <owl:Class rdf:about="#swtichOn"/>
26            </owl:unionOf>
```



```
27            </owl:Class>
28          </owl:allValuesFrom>
29       </owl:Restriction>
30    </rdfs:subClassOf>
31 </owl:Class>
```

Figure 4. Code Extract of a Smart Lamp and its LampController Relationship

5.2. The Discovery Matching Process Explained

As shown in Figure 2, the process execution engine not only needs to have a precise idea of the involved services, but also has to be able to link them with the associated entities (through their actuators and/or sensors). This operation is performed in 3 steps:

1. The ontology is instantiated for a given real smart home.
2. When actuators and sensors are installed in a smart home, they publish the address of their WSDL files and semantic description documents in the service registry.
3. Upon starting, the execution engine invokes the discovery engine, which finds all the necessary services and matches them with their corresponding entities (see Figure 5). To perform this task, it uses an ontology inference mechanism (*e.g.*, SPARQL query language [26]).

```
run:
PREFIX  diuf: <http://www.unifr.ch/owl/2011/03/HomeOntology.owl#>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT  ?h
WHERE
  { ?h  rdf:type          diuf:Heater ;
        diuf:hasActuator  ?a .
    ?a  rdf:type          diuf:HeaterController .
  }

-------------------------------------------------------------------
| h                                                               |
===================================================================
| <http://www.unifr.ch/owl/2011/03/HomeOntology.owl#Heater_61> |
| <http://www.unifr.ch/owl/2011/03/HomeOntology.owl#Heater_62> |
| <http://www.unifr.ch/owl/2011/03/HomeOntology.owl#Heater_60> |
-------------------------------------------------------------------
```

Figure 5. Example of a SPARQL Query to find all Smart Heater Instances

6. The GPL4SE[1] Domain Specific Language

Among the approaches used for programming devices, [27] proposes a macro language that extends Java in order to help developers with creating context-aware applications. [28] proposes an agent-based Domain Specific Language for controlling appliances and providing a single interface for each device. However, it does not efficiently support the management of events. Compared to composition languages like BPEL4WS and BPMN, our DSL supports a high degree of

[1] Generic Process Language for Smart Environments

abstraction and hides several programming details. It allows users to define processes in a more compact and comprehensible way. It provides interaction ability with final users and generic declaration of involved services using a domain ontology rather than Web Service interface descriptions. This language is called GPL4SE (Generic Process Language for Smart Environments) and it is the link among the different elements of our framework (*i.e.*, ontology, service discovery and execution engine). The basic concepts and the global philosophy of GPL4SE are explained in the next section. We then illustrate its use through the energy saving scenario of Section 3.

6.1. GPL4SE Program Structure and Basic Concepts

In this section, we describe the structure of a program written in the GPL4SE language. The program has two main parts: a declaration section and an execution section. The elements of these two sections are described in Figure 6.

Now that we have seen the main structure of a GPL4SE program, we can present some of its principal constructs:

- ***Control flow Constructs***: they define the program's procedural logic. The important ones are: *for*, *while* and *if*.
- ***Execution Constructs***: Execution can be sequential or parallel by using the keywords *sequence* and *flow*, respectively. Other execution constructs, inspired from BPEL, are also provided, such as the *pick* construct.
- ***Service Interaction Constructs***: They allow to interact with devices and there are three types of them: (i) an *action* construct is used to invoke a service, which modifies the state of the corresponding entity; (ii) a *query* construct is used to invoke a service, which retrieves the current state of the corresponding entity; (iii) an *OnEvent* construct allows the process to wait for an event from a publisher or an alarm (*e.g.*, an expiration of a timeout). The messages passed to and from the process for all these constructs are in a Map collection form (*i.e.*, a list of (name, value) pairs).

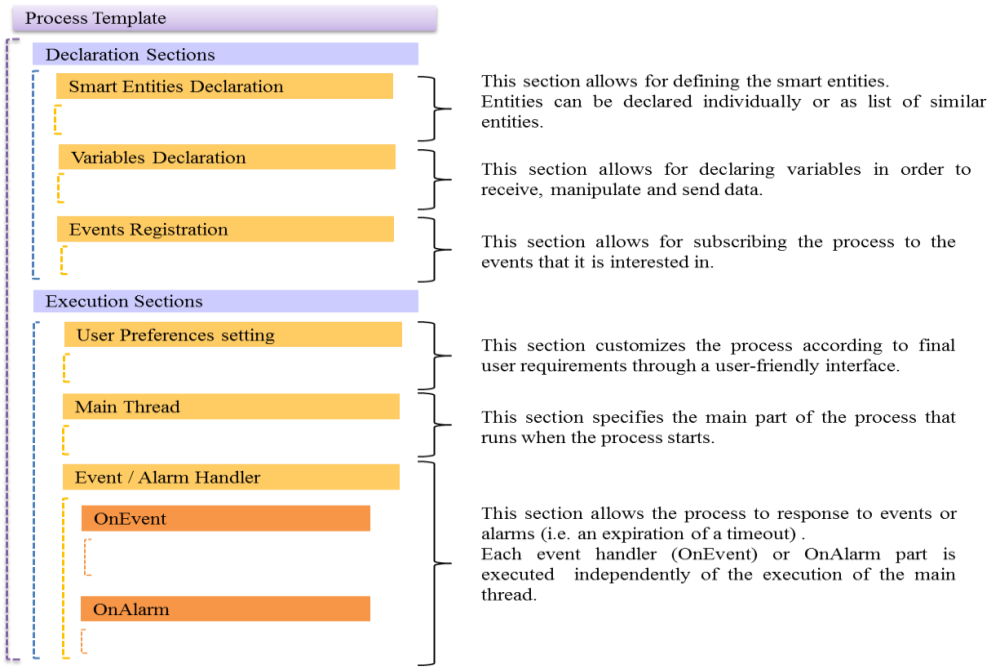

Figure 6. GPL4SE Program Structure

- ***User Preferences' Constructs***: There are two types of these constructs. The *print* construct is used to display information and the *set* construct is used to assign user preferences.

6.2. Motivating Scenario Implementation

Let us now implement the energy saving scenario of Section 3 to illustrate the concepts of the GPL4SE language. Figure 7 shows an extract of this implementation and the next points describe it:

- *Smart Entities Declarations*: lines 3-5 specify the involved smart entities. This allows for controlling them through their associated services.
- *Variables Declarations:* line 8 declares a variable.
- *Events Registration*: lines 11-12 register the process to two events.
- *User Preferences setting*: lines 14-19 get and set user preferences.
- *The main Thread*: lines 20-33 specify the main body of the process.
- *Event/ Alarm Handlers*: line 35 specifies an *OnEvent* handler.
- *Control flow and Execution Constructs*: lines 21-33 give an example of the use of *sequence, while, pick, if,* and *for* constructs.
- *Service Interaction Constructs:* line 27 shows how the action construct invokes the "switchOff" service of the Heater smart entity.
- *Map Data Structures:* in lines 24 and 35, two Map data structures (*accessInput* and *showerInput*) are declared. They contain (name, value) pairs according to the defined ontology. For example, in lines 36 and 39, the values of the *faucetState* and the *locationID* are accessed through getters.

7. Implementation

In order to execute our language in the framework described in Section 5, we have built a translator to convert the process templates into executable processes using a list of rules (a grammar) (see Figure 8). In fact, we generate a recursive-descent parser that reads our templates and produces BPEL4WS and OWL-S executable statements. We have also built the Service Discovery, which forms another part of the framework, using the proposed ontology. For the smart home scenario, the devices are simulated and described in Web Service Interfaces. Each device (actuator or sensor) has one interface that provides its services (queries, events and actions). The evolution of service invocations and the produced events can be visualized in a graphical user interface. Figure 9 shows a snapshot of this interface. The bottom part allows for artificially publishing various events such as Calendar Event or Shower Event. The upper part shows the reaction of the system (Smart Home) according to the action scenarios.

```
1   Scenario template{
2   // Smart Entities Declaration
3   List heaters = discover(Heater);
4   List lamps   = discover(Lamp);
5   List showers = discover(Shower);
6   ...
```

```
7   // Variables Declaration
8   int showerDuration;
9   ...
10  // Events Registration
11  subscribe AccessEvent,
12  subscribe ShowerEvent;
13  ...
14  // User Preferences setting
15  Preferences{
16    print("Give Shower Max duration?");
17    set(showerDuration);
18   ...
19  }
20  // main thread
21  sequence {
22   while (true){
23     pick{
24       OnEvent(AccessEvent, Map accessInput){
25         if(accessInput.get("presenceState") == false){
26           for (Heater heater : heaters){
27              action (heater, "switchOff");}
28           for (Lamp lamp : lamps){
29              action (lamp, "switchOff");}
30       }}//end OnEvent
31       ...
```



```
32      }// end pick
33  }}//  end while/sequence
34  // Event & Alarm Handlers
35  OnEvent(ShowerEvent, Map showerInput){
36   if(showerInput.get("faucetState") == "open"){
37     wait(showerDuration * 60); // in second
38     for (Shower shower : showers){
39       if(shower.locationID == showerInput.get("locationID"){
40         action (shower, "close");
41  }}}}//end OnEvent
42  ...
43  } // end scenario
```

Figure 7. Code Extract of the Energy Saving Scenario

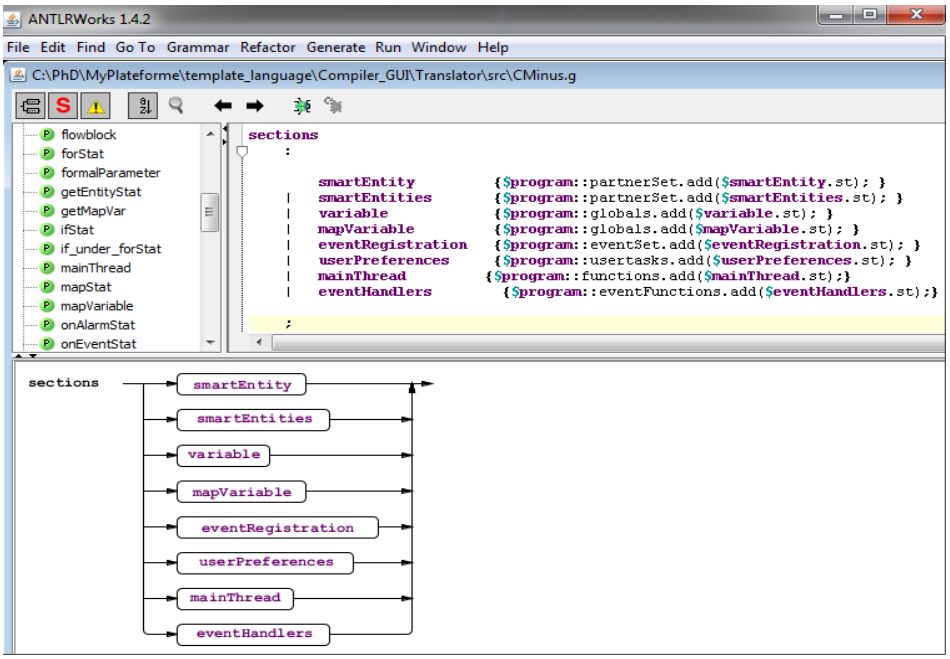

Figure 8. Grammar Development Environment for Building the GPL4SE Translator

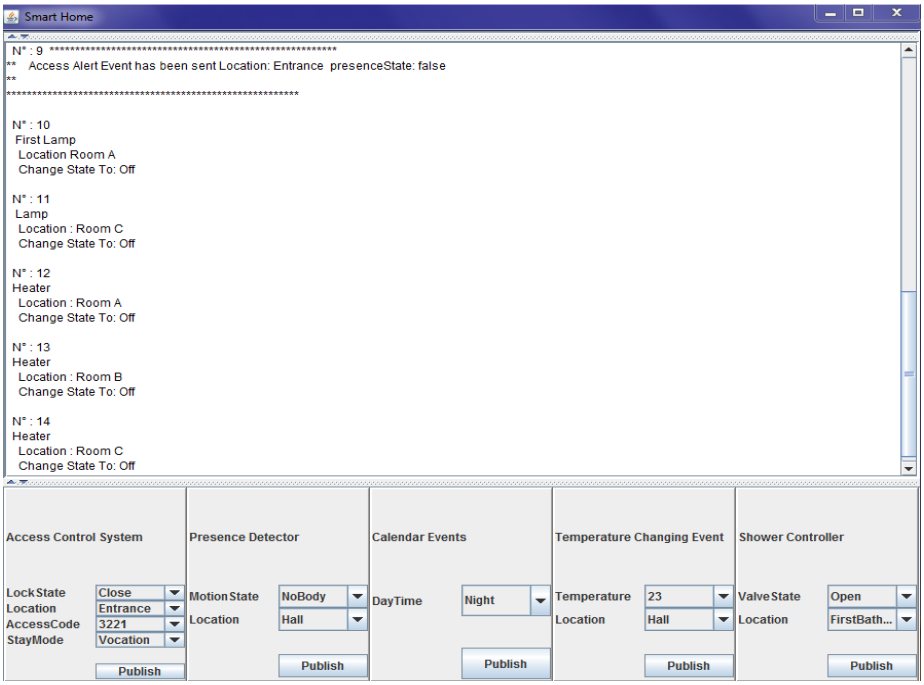

Figure 9. Smart Home Interface for Controlling Services Invocations

8. Conclusion

In In this paper, we propose a service composition framework that facilitates the definition and the execution of complex scenarios in the context of smart residential environments. Its contributions to the research community are twofold:

1. The first one is the generic software architecture presented in Figure 2. The attentive reader should notice that it is a pluggable architecture based on actual state of the arts standards: the services proposed by the various sensors and actuators are described using WSDL and OWL-S; the chosen process templates are transparently translated into BPEL4WS for execution; and the ontology, which serves as the glue between the entities of the physical world and the services of the virtual one, is based on OWL. In other words, we do not reinvent the wheel, but we offer a modular software solution to cleanly integrate proven technologies for modeling and controlling at a high level of abstraction smart residential environments. To be applied, our solution only require two reasonable conditions as described in Section 5: (1) the existence of an appropriate ontology for the chosen environment (home, hospital, school, etc.) and (2) the existence for each service provided by the concrete actuators and sensors of a declarative description using the exact same terminology as the ontology.

2. The second contribution is the GPL4SE domain specific language presented in Section 6. As shown in Figure 5 by its well defined general structure and by the energy saving code extract of Figure 6, this language is divided into two distinct parts: First the declaration section uses the ontology vocabulary to transparently discover the entities of the smart environment and their associated services (events, queries and actions). Secondly, the execution section offers in a simplified manner all the essential control structures of a well-established process control language such as BPEL4WS. In that sense, GPL4SE represents the indispensable complement to our proposed software architecture. It is the language, which describes our scenarios using a precise vocabulary without having to program intricacies such as services grounding, and which is then transparently translated into a standard process language (*e.g.*, BPEL4WS) for execution.

To validate our approach, we have implemented and tested a simple energy saving scenario. We are actually actively exploring the challenges proposed by more complex scenarios, which might include conflicting goals such as optimizing energy, comfort and security simultaneously.

References

[1] F. Sadri, "Ambient Intelligence, A Survey", ACM Computing Surveys (CSUR), New York, **(2011)**, pp. 36:1-36:66.
[2] G. Stavropoulos Thanos, V. Dimitris and V. Ioannis, "A survey of service composition in ambient intelligence environments", Artificial Intelligence Review Journal, Springer, Netherlands, **(2011)**.
[3] M. P. Papazoglou, "Service -Oriented Computing: Concepts, Characteristics and Directions", Proceedings of the Fourth International Conference on Web Information Systems Engineering, IEEE Computer Society, Washington, **(2003)**.
[4] W3C, "OWL 2Web Ontology Language", http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/ [Accessed 14.11.2013].
[5] ZigBee, www.zigbee.org [Accessed 14.11.2013].
[6] Open Services Gateway initiative (OSGi), http://www.osgi.org [Accessed 18.09.2013].
[7] S. Thanos, K. G., V. Dimitris and V. Ioannis, "aWESoME: A web service middleware for ambient intelligence", Expert Systems with Applications, Pergamon Press Inc, Tarrytown, **(2013)**, pp. 4380-4392.
[8] R. Chander, "Novel Ubiquitous Interoperable Context-aware Smart Environments through Web Services", Proceedings of the International MultiConference of Engineers and Computer Scientists, Hong Kong, **(2012)**.
[9] P. Antonio, C. Davide, P. Andrea and G. Alessandro, "Connecting smart things through web services orchestrations", ICWE'10 Proceedings of the 10th international conference on Current trends in web engineering, Springer-Verlag, Berlin, **(2010)**, pp. 431-441.
[10] OASIS, "Web Services Business Process Execution Language Version 2.0", http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html [Accessed 14.11.2013].
[11] OMG, "Business Process Model and Notation", http://www.omg.org/spec/BPMN/2.0/ [Accessed 16.09.2013].
[12] R. De Masellis, C. Di Ciccio, M. Mecella and F. Patrizi, "Smart Home Planning Programs", 7th International Conference on Service Systems and Service Management, Tokyo, **(2010)**, pp. 1-6.
[13] K. Eirini, W. Ehasan, B. Jaap, L. Alexander and A. Marco, "Interoperation, Composition and Simulation of Services at Home", 8th International Conference, ICSOC, Springer-Verlag, Berlin, **(2010)**, pp. 167-181.
[14] C. Davide, C. Berardina, F. Stefano and N. Nicole, "Smart Composition of Services in Situation-Aware Home Environments", 2nd International Workshop on User Modeling and Adaptation for Daily Routines (UMADR), Girona, **(2011)**, pp. 5-12.
[15] W. Paul and K. Dimitrios, "A Framework for End-User Programming of Smart Homes Using Mobile Devices", Consumer Communications and Networking Conference, IEEE, Las Vegas, **(2007)**, pp. 716-721.
[16] C. Di Ciccio and M. Massimo, "The homes of tomorrow: service composition and advanced user interfaces", ICST Transactions on Ambient Systems Journal, **(2011)**, pp. 1-13.
[17] P. Baglietto, M. Maresca, M. Stecca and C. Moiso, "Smart Object Cooperation through Service Composition", 15th International Conference on Intelligence in Next Generation Networks, Berlin, **(2011)**, pp. 133-138.
[18] W. Feng and J. Kenneth, "An Ontology-based Actuator Discovery and Invocation Framework in Home Care Systems", 7th International Conference on Smart Homes and Health Telematics, Springer, Berlin, **(2009)**, pp. 66-73.
[19] S. Nenad and X. Yongchun, "An Intelligent Event-driven Approach for Efficient Energy Consumption in Commercial Buildings", Proceedings of the 5th ACM International Conference on Distributed Event-based System, ACM, New York, **(2011)**, pp. 303-312.
[20] I. Noha and M. Frédéric, "A Survey on Service Composition Middleware in Pervasive Environments", IJCSI International Journal of Computer Science, **(2009)**, pp. 1-12.
[21] F. Gerhard and G. Reinhard, "UML2 activity diagram based programming of wireless sensor networks", proceedings of the ICSE Workshop on Software Engineering for Sensor Network Applications, ACM, New York, **(2010)**, pp. 8-13.
[22] N. Glombitza, D. Pfisterer and S. Ficher, "Using state machines for a model driven development of web service-based sensor network applications", ICSE Workshop on Software Engineering for Sensor Network Applications, ACM, New York, **(2010)**, pp. 2-7.
[23] ORACLE, "Oracle Business Process Manager", http://www.oracle.com/technetwork/middleware/bpel/overview [Accessed 14.11.2013].
[24] W3C, "Web Services Description Language", http://www.w3.org/TR/wsdl [Accessed 14.11.2013].

[25] W3C, "OWL-S: Semantic Markup for Web Services", http://www.w3.org/Submission/OWL-S [Accessed 17.09.2013].
[26] W3C, "SPARQL Query Language for RDF", http://www.w3.org/TR/rdf-sparql-query/ [Accessed 14.11.2013].
[27] S. Sanjin, L. Fei and D. Schahram, "COPAL: A Macro Language for Rapid Development of Context-aware Applications in Wireless Sensor Networks", Proceedings of the 2nd Workshop on Software Engineering for Sensor Network Applications, ACM, New York, **(2011)**, pp. 1-6.
[28] D. Adolf, E. Ferranti and S. Koch, "SmartScript-A Domain-Specific Language for Appliance Control in Smart Grids", 3th International Conference On SmartGrid Communications, Tainan, **(2012)**, pp. 465-470.

Authors

Abdaladhem Albreshne is a PhD Student and a member of the Software Engineering Group in the Department of Informatics of the University of Fribourg-CH. He received his Master degree in Communication and Computer Networks from the Institute Telecom SudParis, France. His Ongoing research topics focus on Smart Home Environments; Ontology; Semantic Language, Web Service, Web Service Discovery and Service Oriented Architecture.

Ayoub Ait Lahcen is an assistant professor at the ENSA-Kenitra, a Moroccan engineering school, and a researcher at the Laboratory for Research in Computing and Telecommunications (LRIT), which is located in the Faculty of Sciences of Rabat. Prior to that, he got a Swiss Government Scholarship in order to work, as a postdoctoral researcher, during the academic year 2012-2013 in the Software Engineering Group of the University of Fribourg-CH. His doctoral thesis has been prepared in joint guardianship with Nice Sophia Antipolis University (at INRIA Sophia Antipolis Research Center) and Mohammed V Agdal University (at LRIT laboratory). His research interests include Component-Based Software Development, Service-Oriented Architecture, Peer-to-Peer and service interactions in smart environments.

Jacques Pasquier holds the software engineering chair at the Department of Informatics of the University of Fribourg-CH, where he is currently Vice Rector responsible for the IT infrastructures and international relationships. Before joining the DIUF in 1987, he was a senior research associate at the T.J. Watson Research Center in Yorktown Heights, USA and a Swiss NSF scholarship student at the Mathematical Sciences Department of the Johns Hopkins University in Baltimore, USA. Professor Pasquier current research interests include the design of Service Oriented Architectures (SOA) with a special emphasis on RESTful web services, the Web of Things (WoT), as well as the composition of context aware web services in smart homes environments. His prior research activities focused on modeling complex reliability systems, software patterns and frameworks, hypermedia and new learning technologies, as well as distributed multi-agents systems. He is the author or co-author of three books and around thirty refereed papers.

