Pervasive Service Model: Using OWL-S to extend Service Model in OSGi Service Platform

Chi-Wei Huang, Kuen-Min Lee
Industrial Technology Research Institute, Taiwan
{ireta_huang, allen_lee}@itri.org.tw

摘要

本篇論文提出一個新的服務模組:普遍性服務模組 (PSM), 它主要使用 OWL-based Web Service Ontology (OWL-S) 的方法來改善原先在 Open Services Gateway Initiative (OSGi) 下的服務模組. 這個新服務模組的最大特色就是, 服務提供者可以透過它來跟家庭裡面目前正在運作的服務溝通, 進而開發出整合性的新服務. 同時它也允許使用者可以自行組合家庭裡的服務成為一個工作項目來執行或者是排程定時執行. 除了系統預設的使用情境之外, 使用者也可以依據本身的需求來修改這些使用情境, 或是自行開發一個新的使用情境. PSM 的目的就是希望家庭裡的服務平台是能夠真正的以人為中心來設計, 讓使用者可以很彈性, 很方便, 很隨性的使用家裡面現有的服務, 而不是去遷就系統所提供的使用方式, 而改變自己的使用習慣.

關鍵詞：OWL-S, OSGi, Pervasive Service, Scenario.

Abstract

This paper proposes an approach which is using OWL-based Web Service Ontology (OWL-S) to extend the original service model in the framework of Open Services Gateway Initiative (OSGi) as a new model called Pervasive Service Model (PSM). This approach can help service providers to develop integrated services by negotiating with various home services. It also allows user to execute and schedule daily tasks by using or combining existed home appliances and services. Except the system default scenarios, user now can develop and modify user-defined scenarios based on needs. What PSM will achieve is such a human-centric, flexible and integrated rich services environment applying to home.

Keywords: OWL-S, OSGi, Pervasive Service, Scenario.

1. Introduction

For information integration, family is quite a special and complicated environment as compared with office and factory. The whole infrastructure for all devices, network and communication interface can not be clearly defined at the beginning. When the demand appears, various kinds of appliances or services will just enter the family one by one. For example, security system is needed when theft-proof requirement is appeared, and health-care system is needed if there are elders in the family needed to look after… etc. As a result, when family becomes a rich environment with various services flooded in, making these services to be reusable, produce more add-on values, and customize based on every family needs will be a big issue. The OSGi Service Platform specification delivers an open, common architecture for developers and service providers to develop, deploy and manage services in home networks and devices [1]. Each service in OSGi is not an independent system anymore. Services can cooperate inner or cross this service platform. But the shortage is this communication behavior is programmed, predefined and not changeable. It is still not a human-centric designed, but somehow the most important role in the family is people. No matter how strong the system is, it can't be applied to the whole family in order to satisfy everybody's requirements. Thus a good design is not to solve all requirements, but a way to let user used to solve all what he needs. There should be a mechanism, supplied by an open service platform, to let user can develop his own system and work flow based on his requirement by using and compositing existed services in home.

This paper presents Pervasive Service Model (PSM) that fulfills these requirements. In what follows, we will discuss some background and technologies on session two, then propose a new model, PSM, to solve all problems on session three which includes the model design, management services and an implementation example, and finally make a conclusion on session four.

2. Background

This section will briefly describe related background technologies and other researchers' works in order to solve the service integration and adaptation issue which is highlighted in the abstract and introduction sections.

2.1 The Service Model in OSGi

OSGi is a kind of middleware. Each application in OSGi framework is called bundle. The OSGi's Service Model defines a dynamic collaborative model that bundles can register services, search for them, receive notifications when their registration state changes, or use other registered services [1]. It has the following essentials:

- **Collaborative**: The service layer must provide a mechanism for bundles to publish, find, and bind to each other's services without having a priori knowledge of those bundles.
- **Dynamic**: The service mechanism must be able to handle changes in the outside world and underlying structures directly.
- **Secure**: It must be possible to restrict access to services.
- **Reflective**: Provide full access to the Service Layer's internal state.
- **Versioning**: Provide mechanisms that make it possible to handle the fact that bundles and their services evolve over time.
- **Persistent Identifier**: Provide a means for bundles to track services across Framework restarts.

This Service Model is well defined and mature enough for using, but it still has shortage of a service in OSGi is just a normal Java object, and the necessary step of service registration is depended on one or more Java interfaces. If there is a service implemented by none Java technology or existed on another services platform, it can't access services registered in OSGi.

2.2 OWL-S: Semantic Markup for Web Services

OWL-S [3] is an ontology language, within the OWL–based [2] framework of the Semantic Web, for describing Web services. It will enable users and software agents to automatically discover, invoke, compose, and monitor Web resources offering services, under specified constraints.

In this paper, OWL-S is used not only on web service, but also on UPnP services, Java services… etc. OWL-S will be a kind of common language between various home services.

2.3 UPnP: Universal Plug and Play

UPnP [4] is a set of computer network protocols which allow devices to connect seamlessly and to simplify the implementation of networks in the home (data sharing, communications, and entertainment) and corporate environments. As for services, it is the most adopted solution for integration technology in home if services can also plug and play for users.

2.4 MONET: Mathematics on the Net

The aim of the MONET is to create a semantic web to the world of mathematical software, using sophisticated algorithms to match the characteristics of a problem to the advertised capabilities of available services and then invoking the chosen services through a standard mechanism. The resulting framework will be powerful, flexible and dynamic, yet robust and easy to navigate, putting state-of-the-art algorithms at the disposal of users anywhere in the world [5].

2.5 Task Computing

Task computing is a new paradigm for how users interact with devices and services. It allows users to focus on the tasks they want to accomplish with computing devices rather than how to accomplish them [6]. It aims to empower non-expert users with the ability to perform complex tasks in information-rich, device-rich, and service-rich environments. To achieve this, it exposes the functionality in such environments as Semantic Web services, which in turn the user can discover and arbitrarily compose [7]. It also combines UPnP for Semantic Service Discovery Mechanism, OWL-S for the semantic service description, and UPnP and Web Services for service invocations.

3. Pervasive Service Model

This paper proposes a new model called Pervasive Service Model (PSM) which enhances the original OSGi Service Model to let services not only limited on Java-based services but all home services. Pervasive service means a common, widespread, intuitional home service which includes UPnP service, web service, and original OSGi service… etc. Due to home is such a rich environment which is flooded with various and heterogeneous services, PSM is needed for managing and using all kinds of pervasive services. To make use of all pervasive services, a computer-interpretable description for pervasive service is needed. That's OWL-S, a Semantic Web markup language. The deference is OWL-S in PSM is not only used for web services but all services in home, pervasive services, for service declaration and description. It also proposes a mechanism to let pervasive services can perform registration, invocation and cooperation with each other.

3.1 Essentials

The PSM has following essentials based on the design of OWL-S ontology and OSGi Service Model:

- **Registry Management**: Provide a mechanism for service discovery, registration, un-registration and removing.

- **Invoke Management**: Provide a mechanism to search services, retrieve service profile [3] and service model [3], and execute service as a collection of remote procedure calls. OWL-S provides a declarative, computer-interpretable API that includes the semantics of the arguments to be specified when executing these calls, and the semantics of that is returned in messages when the services succeed or fail. A software agent should be able to interpret this markup to understand what input is necessary to invoke the service, and what information will be returned.
- **Composition and Interoperation Management**: Involve the selection, composition, and interoperation of pervasive services to perform some complex or customized task. With OWL-S, software can be written to manipulate these declarative specifications of the prerequisites and consequences of individual services, service compositions and data flow interactions. Then execute target task automatically.
- **Authorization Management**: Provide access privileges for pervasive services, home users, service providers, and system integration consulters… etc. This mechanism also includes the limitation of pervasive service's visibility and usability… etc.

3.2 Entities

PSM itself is a management pervasive service which is designed to manage pervasive services. It is also a Composite Process [3] composed by three Atomic Processes [3]: Registry Manager Process, Event Handler Process and Scenario Invoker Process, and a set of bundle components as figure 1 shows.

- **Registry Manager Process**: A facility to process management request for pervasive services and return request results.
- **Event Handler Process**: A facility to process event registration and dispatch for pervasive services.
- **Scenario Invoker Process**: A facility to invoke system defined or custom defined scenario.
- **Service Management Component**: A facility to let Registry Manager Process register and un-register service, search and retrieve service profile, and remove service.
- **Service Registry Component**: Process the pervasive service's registration includes registering, un-registering, and searching services.
- **Service Profile Component**: Holds and maintains the pervasive service's service profile.
- **Service Permission Component**: Holds the permission settings for invoking a pervasive service and retrieving pervasive service profile.
- **Service Filter Component**: An object that implements a simple but powerful filter language. It can select on properties.
- **Auto-Discover Component**: A process for performing automated discovery to let Service Management component can manipulate service registry.
- **Service Event Dispatch Component**: An event holding information about the registration, notification, or un-registration of a pervasive service. This component can perform service listener registration, and event receiving and dispatching.
- **Scenario Management Component**: A facility to let Scenario Invoker Process invoke existed scenarios, retrieve scenarios description, add or modify or remove scenarios, and scheduling or executing scenario.

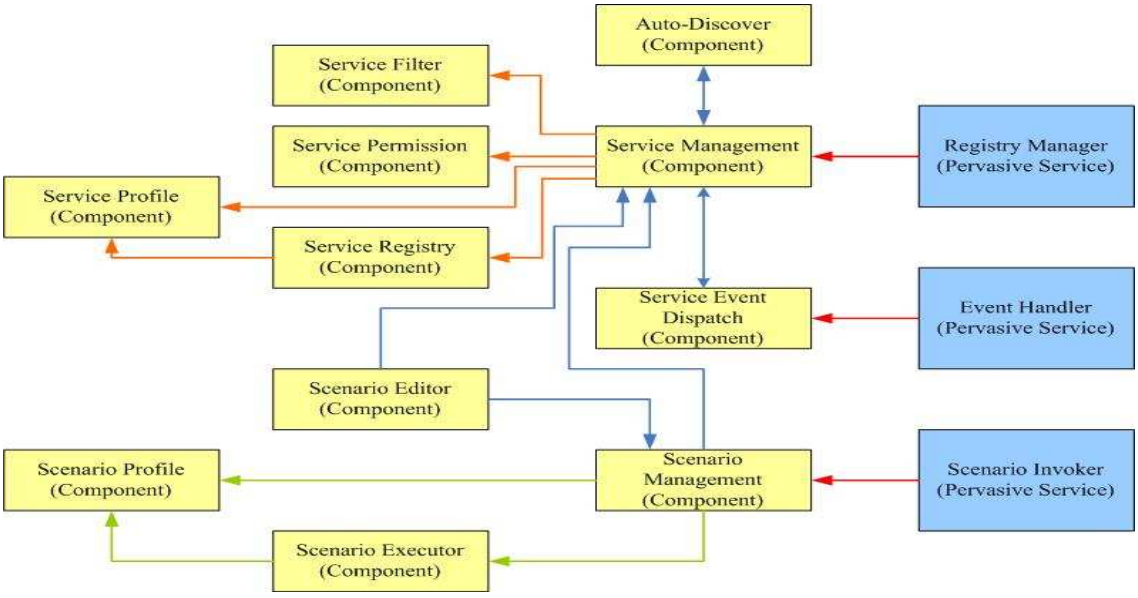

Figure 1: Component Diagram for PSM

- **Scenario Profile Component**: Holds and maintains the scenario's description profile.
- **Scenario Editor Component**: An UI component to let user can compose pervasive services as a scenario based on what they want to do within PSM.
- **Scenario Executor Component**: A process for executing or scheduling scenarios.

3.3 Service Management

Basically, PSM uses OSGi as an open service platform to integrate all kinds of network, services and appliances in home. Every service follows PSM's framework is called a pervasive service and must have a service profile registered for service description which includes characteristics, functionalities, messages, and communication protocol. Service management is such a mechanism which manages and maintains all service profiles, permission configurations, and service registry in PSM. It provides an interface to discover, search, register, un-register pervasive services and retrieve service profile based on permission settings.

3.3.1 Service Invocation

In OWL-S, it defined a method, Service Grounding [3], to specify the details of how to access the service with protocol and message formats, serialization, transport, and addressing. It also suggests using Web Services Description Language (WSDL) as a grounding mechanism. By the help of OSGi, PSM now is not limited on choosing WSDL as the only grounding. It can have lots of candidate protocol like SOAP, UPnP as grounding mechanism because OSGi integrated it and made it happened.

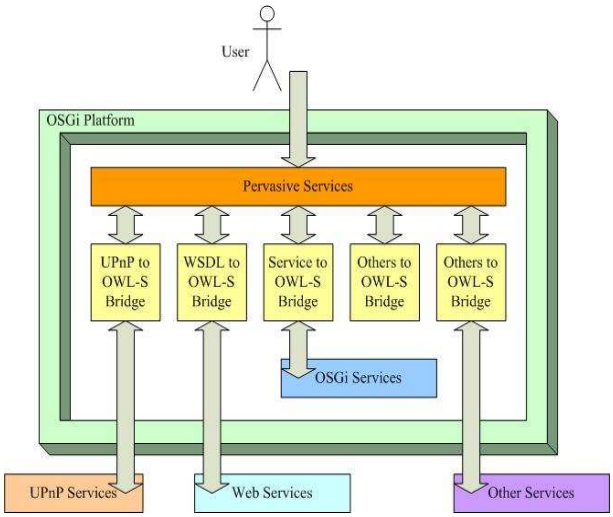

Figure 2: Service Mapping Diagram

3.3.2 Bridge Bundle

In order to let PSM to cover all kinds of services in home network, corresponding bridge bundle is needed for mapping or translating original service to pervasive service. As figure 2 shows, target mapping service can be an UPnP service, a web service, an OSGi service, or even a simple Java object. The bridge bundle should be able to construct a service profile, clearly specify the service grounding and register it in PSM. In the other side, this bridge bundle should be familiar with the target communication protocol for executing the request invocation from pervasive services.

3.4 Service Event Handling

In PSM, service events like registration, modification, notification and un-registration should be passing between services. Pervasive service can register target listening event type for particular pervasive service and PSM will take the responsible of event dispatching if the target listening event activated.

3.5 Scenario Management

In OWL-S, a scenario is defined as a composite service which is composed by atomic or composite processes by using control constructs [3]. Scenario is not a behavior a service will do, but a behavior (or set of behaviors) the user can perform by sending and receiving a serious of messages. There are two types of scenario: system defined scenario and custom defined scenario.

3.5.1 System Defined Scenario

System defined scenario is a predefined scenario provided by production supplier or system integration provider. Usually it is bounded with service delivery. It is designed for the usability of production, integrated application for a set of appliance, or a system with particular purpose such as security, automation, healthcare, or entertainment. Home appliance is not to be made and produced for one single purpose. It can play different rules in different scenario now for more add-on values by interacting and cooperating with other appliances.

3.5.2 User Defined Scenario

The main purpose of user defined scenario is that user can properly make use of appliances and services in home based on what he really needs as time goes on. User can modify the existing system defined scenario for special condition and save it as a new one or totally define a new one based on different requirement and purpose. Also user can modify the

existed scenario to apply the changes of new appliance instead of the original or broken old one.

3.5.3 Scenario Editor

One purpose for all PSM's foundations is in order to implement Scenario Editor (SE) to let user can design scenario and execute it. In SE, pervasive services can be combined and cooperate with other pervasive services. As figure 3 shows, SE has a service list on left top which lists all available pervasive services and a property list which lists all related service properties. The main scenario's task flow is showed on center and user can drag-and-drop flow control components presented on left down. A flow control list on right top lists all flow control components of current scenario. User can write expressions or statements related to the target flow control component at statement editing panel on right down.

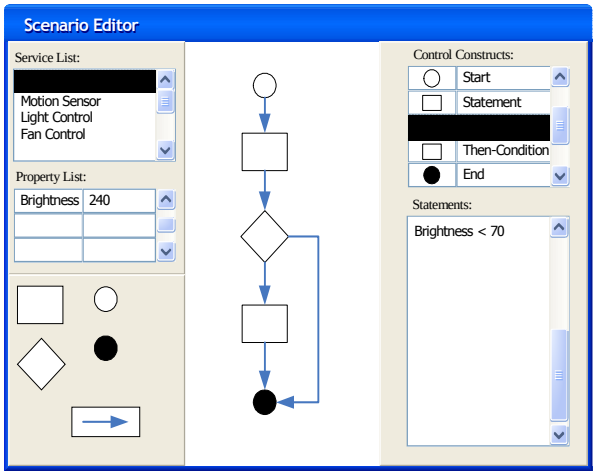

Figure 3: Scenario Editor

Figure 3 also shows a simple task flow example: "Auto- Lighting" scenario that will automatically turn on lamp when the indoor brightness value is lower then 70. Two pervasive services are used in this scenario: "Brightness_Sensor" which provides "GetBrightness" service and "Light_Control" which provides "LightOn" service. The sample service profiles are listed bellow:

Brightness_Sensor_Process.owl

```
<process:AtomicProcess rdf:ID="GetBrightness">
 <process:hasOutput>
  <process:Output ref:ID="Brightness_Value">
   <process:parameterType rdf:resource="&xsd;#integer">
   </process:parameterType>
  </process:Output>
 </process:hasOutput>
</process:AtomicProcess>
```

Brightness_Sensor_Grounding.owl

```
<grounding:WsdlAtomicProcessGrounding rdf:ID=
      "GetBrightnessGrounding">
 <grounding:owlsProcess rdf:resource="#GetBrightness"/>
 <grounding:wsdlOperation>
  <grounding:WsdlOperationRef>
   <grounding:portType rdf:datatype="&xsd;#anyURI">
     #GetBrightness_PortType</grounding:portType>
   <grounding:operation rdf:datatype="&xsd;#anyURI">
     #GetBrightness_operation</grounding:operation>
  </grounding:WsdlOperationRef>
 </grounding:wsdlOperation>

 <grounding:wsdlOutputMessage rdf:datatype="&xsd;#anyURI">
     #GetBrightness_Output</grounding:wsdlOutputMessage>
 <grounding:wsdlOutput>
  <grounding:WsdlOutputMessageMap>
   <grounding:owlsParameter rdf:resource=
        "#Brightness_Value"/>
    <grounding:wsdlMessagePart rdf:datatype="&xsd;#anyURI">
      #brightnessValue</grounding:wsdlMessagePart>
   </grounding:WsdlOutputMessageMap>
 </grounding:wsdlOutput>

 <grounding:wsdlReference rdf:datatype="&xsd;#anyURI">
     http://www.w3.org/TR/2001/NOTE-wsdl-20010315
 </grounding:wsdlReference>
 <grounding:wsdlDocument rdf:datatype="&xsd;#anyURI">
 http://www.itri.org.tw/osgi/psm/BrightnessSensorGrounding.wsdl
 </grounding:wsdlDocument>
</grounding:WsdlAtomicProcessGrounding>
```

Light_Control_Process.owl

```
<process:AtomicProcess rdf:ID="LightOn">
 <process:hasInput>
  <process:Input rdf:ID="Lamp_ID">
   <process:parameterType rdf:datatype="&xsd;#anyURI">
     &xsd;#string
   </process:parameterType>
  </process:Input>
 </process:hasInput>
</process:AtomicProcess>
```

Light_Control_Grounding.owl

```
<grounding:WsdlAtomicProcessGrounding
  rdf:ID="LightOnGrounding">
 <grounding:owlsProcess rdf:resource="#LightOn"/>
 <grounding:wsdlOperation>
  <grounding:WsdlOperationRef>
   <grounding:portType rdf:datatype="&xsd;#anyURI">
     #LightOn_PortType</grounding:portType>
   <grounding:operation rdf:datatype="&xsd;#anyURI">
     #LightOn_operation</grounding:operation>
  </grounding:WsdlOperationRef>
 </grounding:wsdlOperation>

 <grounding:wsdlInputMessage rdf:datatype="&xsd;#anyURI">
     #LightOn_Input</grounding:wsdlInputMessage>
 <grounding:wsdlInput>
  <grounding:WsdlInputMessageMap>
   <grounding:owlsParameter rdf:resource="#Lamp_ID"/>
   <grounding:wsdlMessagePart rdf:datatype="&xsd;#anyURI">
     #lampID</grounding:wsdlMessagePart>
```

```
  </grounding:WsdlInputMessageMap>
 </grounding:wsdlInput>

 <grounding:wsdlReference rdf:datatype="&xsd;#anyURI">
     http://www.w3.org/TR/2001/NOTE-wsdl-20010315
 </grounding:wsdlReference>
 <grounding:wsdlDocument rdf:datatype="&xsd;#anyURI">
     http://www.itri.org.tw/osgi/psm/LightControlGrounding.wsdl
 </grounding:wsdlDocument>
</grounding:WsdlAtomicProcessGrounding>
```

The abstract description of this task flow is: first get brightness value from brightness sensor, compare this brightness value with 70, and turn on lamp if the brightness value is lower then 70. SWRL: A Semantic Web Rule Language [9] is used for rules presentation. The result scenario profile is listed bellow:

Auto-Lighting_Scenario.owl

```
<process:CompositeProcess rdf:ID="AutoLightScenario">
 <process:composedOf>
  <process:Sequence>
   <process:components>
    <process:ControlConstructList>
     <objList:first>
      <process:Perform rdf:ID="GetBrightnessPerform">
       <process:process rdf:resource="#GetBrightness"/>
        <process:hasDataFrom>
         <process:InputBinding>
          <process:valueSource>
           <process:ValueOf>
            <process:theVar rdf:resource=
                 "#Brightness_Value"/>
            <process:fromProcess rdf:resource=
                 "&process;#TheParentPerform"/>
           </process:ValueOf>
          </process:valueSource>
         </process:InputBinding>
        </process:hasDataFrom>
       </process:Perform>
      </objList:first>
      <objList:rest>
       <process:ControlConstructList>
        <objList:first>
         <process:If-Then-Else>
          <process:ifCondition>
           <expr:SWRL-Condition rdf:ID=
                "IsInLowBrightness">
            <expr:expressionBody rdf:parseType="Literal">
             <swrl:AtomList>
              <rdf:first>
               <swrl:DataRangeAtom>
                <owl:OneOf>
                 <owl:DataValue owl:datatype="&xsd;int">
                      0</owl:DataValue>
                 <owl:DataValue owl:datatype="&xsd;int">
                      70</owl:DataValue>
                </owl:OneOf>

                <ruleml:var>#Brightness_Value</ruleml:var>
               </swrl:DataRangeAtom>
              </rdf:first>
                <rdf:rest rdf:resource="&rdf;#nil"/>
              </swrl:AtomList>
             </expr:expressionBody>
            </expr:SWRL-Condition>
           </process:ifCondition>
           <process:then>
            <process:Perform rdf:ID="LightOnPerform">
             <process:process rdf:resource="#LightOn"/>
           </process:then>
          </process:If-Then-Else>
         </objList:first>
         <objList:rest rdf:resource="&objList;#nil"/>
        </process:ControlConstructList>
       </objList:rest>
      </process:ControlConstructList>
     </process:components>
    </process:Sequence>
   </process:composedOf>
</process:CompositeProcess>
```

4. Conclusions and Future Work

The idea of PSM is to achieve the design of information integration system should be human-centric and developed by user's needs. Follow the PSM, service provider can focus on home service development and PSM will be the communication bridge between user and services. In the future, we hope PSM can analyze user's requires and requests, give service recommendation, and even execute it automatically without user confirmed. In Industrial Technology Research Institute (ITRI), we have already developed a prototype system based on OSGi Service Platform which integrated home security, home automation, health care, and multimedia services. The next step will be to add PSM and implement it in this integrated system and translate all demonstrative scenarios. Then the user will be able to use SE to develop his own scenario and execute it instead of using system defined scenario.

5. References

[1] OSGi http://www.osgi.org
[2] OWL http://www.w3.org/TR/owl-features
[3] OWL-S http://www.w3.org/Submission/OWL-S
[4] UPnP http://www.upnp.org
[5] MONET http://monet.nag.co.uk/cocoon/monet/index.html
[6] Task Computing http://taskcomputing.org
[7] Ryusuke Masuoka, Bijan Parsia, Yannis Labrou and Evren Sirin, "Ontology-Enabled Pervasive Computing Applications," IEEE Intelligent Systems, vol. 18, no. 5, Sep./Oct. 2003, pp. 68-72.
[8] OWL-S 1.1 http://daml.semanticweb.org/services/owl-s
[9] SWRL http://www.w3.org/Submission/SWRL
[10] SWRL 0.6 http://www.daml.org/2004/04/swrl/