Experience of Applying Semantic Web Services to URC

HyeonSung Cho*, YunKoo Chung, Young-Jo Cho
Intelligent Robot Research Division
Electronics and Telecommunications Research Institute
161 Gajeong-dong, Yuseong-gu, Deajeon, Republic of Korea
{ hsc*, ykchung, youngjo}@etri.re.kr

***Abstract* – We have carried out developing a robot which can access internet. This robot, URC (Ubiquitous Robot Companion), is able to get information on the internet and to narrate it in human language. This work was due to the technology of Semantic Web Services, because this technology makes it possible to automate many tasks for machine. This paper presents experience gained during implementation of robot application using Semantic Web Services and shows its possibility and effectiveness in ubiquitous world.**

Keywords – Robot, Semantic Web, Web Services

1. INTRODUCTION

Recently, a number of robotics researches are announced such as Honda's ASIMO and Sony's AIBO. Most of these robots cannot utilize the web contents and have been focused on the issue of robot itself. However, it is important not only robot technologies such as recognizing, walking, navigating, and handling arms but also its services and applications. But useful robot services are still very poor.

Therefore, it is challenging to provide various services and contents from the web. Fortunately, the web already has tremendous contents and services such as weather and traffic information, and reservation services and furthermore its domain is enlarging into the emerging Semantic Web Services [1] and the ubiquitous world.

It is possible to discover and to execute web services automatically in the Semantic Web Services domain. Because the Semantic Web Services is a combination of Semantic Web and Web service technology, which aims at providing means of automatically executing, discovering by annotating web services with semantic mark-up language. Consequently, it is feasible to use these kinds of technologies for the networked robotic services.

In this paper, we present the experience of Intelligent Web Service Framework for the URC (Ubiquitous Robotic Companion) [2] which makes use of the Semantic Web Services. URC is a robotic prototype which is able to get available and rich semantic web services from a URC server through a wireless internet connection. The URC server provides a registry of semantic description of web services and an execution engine for the discovered web services.

2. REQUIREMENTS AND CONSIDERATIONS

There are a lot of issues and requirements in real field to realize that a robot can access the Web. Before implementing the system, we considered many issues and defined the requirements. The basic requirements are that there should be an available instance of Web Services and a robot connected in Internet. These look simple problems.

On this situation, the first encountered issue is a user interface. A user interface means how a user order commands to a robot. It is usually used voice command in robot society. In this case, a system should have the capability of voice recognition. Another problem under voice recognition is to apprehend user intention. The second issue is related to the Semantics of Web Services. A system can find an appropriate Web Services and understand the meaning of it. In Semantic Web world, this problem usually is solved by ontology such as OWL [3] and a service registry such as UDDI [4]. But, UDDI doesn't have enough properties for the semantic processing. The third issue relates execution of a Web Service. It is an easy and simple to solve this problem. It just use a toolkit related on Web Services. However, if the service consists of many Web Services, that is a process of Web Service, it is a difficult problem. The last issue is the presenting the result of Web Service execution. The result format is XML document that is a kind of text. This is usually solved by TTS (Text to Speech) system.

Issues related a robot itself such as recognition, moving control, and navigation is important, but these kinds of subjects do not mention in this paper because these are out of the scope for the focus of this paper. And all issues mentioned above will be treated in details in Section 3.

3. IMPLEMENTATION OF INTELLIGENT WEB SERVICE FRAMEWORK

1). Intelligent Web Service Concept for the URC

We have developed the Intelligent Web Service Framework which includes URC and URC server. URC is an interactive Web terminal which can be any device (e.g. PDA, cellular phone, and notebook) equipped with URC client. URC includes several robotic components such as a voice recognition module, a face recognition module, and a navigation module.

URC server is composed of service planner, semantic service registry and execution engine. In order to provide machine process-able Web services which can be apprehensible by URC, Web services should be described in service ontology, OWL-S [5]. This semantic markup is located in semantic Web services registry while Web services descriptions (that is, WSDL files) are stored other place, which could be service provider or UDDI registry. Service concept is represented in Figure 1.

URC provides robotic services as following steps.

a. Users ask the URC a request.
b. URC analyzes a request using service ontology.
c. URC service planner connects to Semantic Discovery Registry and discovers appropriate services.
d. Most suitable service is executed by URC Process Execution engine.
e. Execution results are returned to URC.

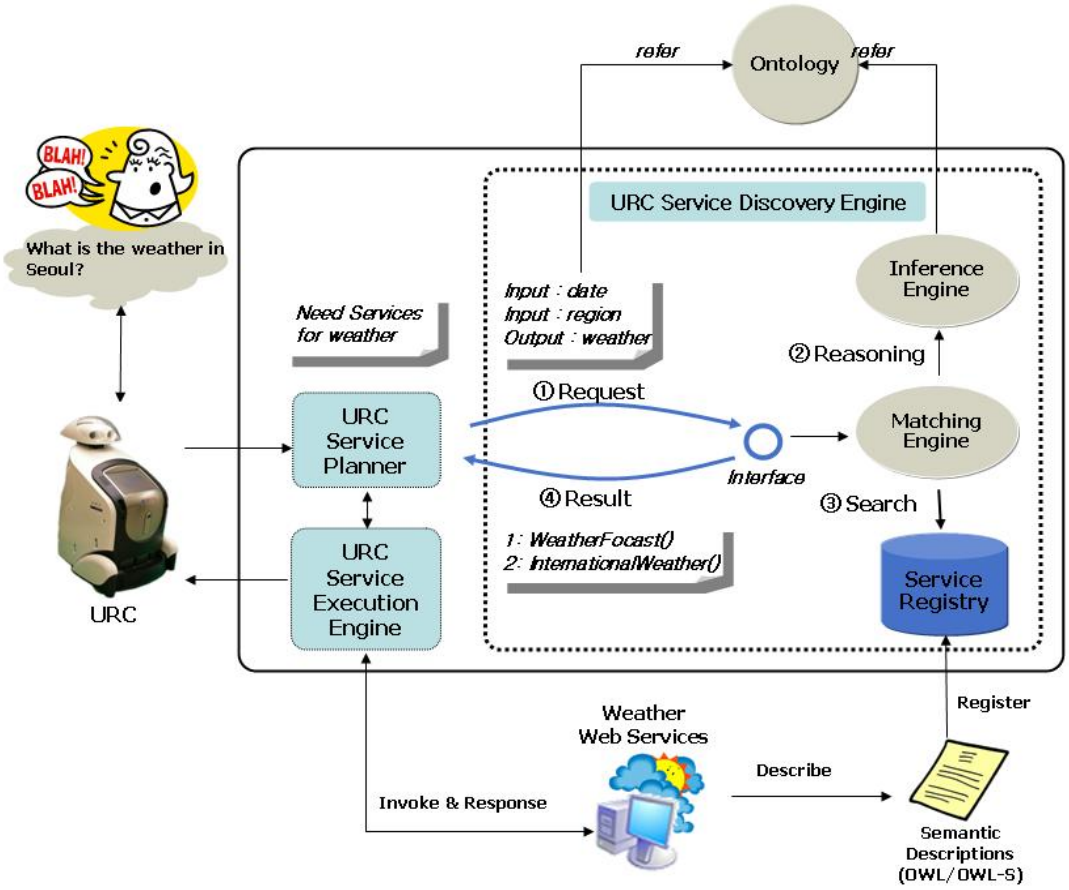

Fig. 1 Intelligent Web Service Concept for the URC

2). Web Services and Semantic Markup

With semantic markup of Web services, we can specify the information necessary for Web service discovery as machine-interpretable semantic markup at the service Web sites and a service registry can automatically locate appropriate services.

OWL-S is a solution of locating Web services on the basis of the capabilities that they provide. To automate Web service discovery, we associate properties with services that are relevant to automated service classification and selection. To automated Web service execution, markup must enable a computer agent to automatically construct and execute a Web service request and interpret and potentially respond to the service's response.

Example of semantic Web service markup about Light Control Service which turns on/off the lamp in the laboratory described in OWL-S is depicted in Figure 2.

3). Service Planning

For performing services, URC Service Planner analyzes the user input which is the formal format from getting the voice recognition process. After analyzing the user intention, it discovers exact services which are requested by URC from URC Registry, and makes a service plan by comparing IOPE (Input, Output, Precondition, and Effect) which is requested by URC with those in OWL-S process model in the URC registry. After discovering services, it generates the invoking scheme as a BEPL4WS [6] format and send it to the web service execution engine. The final mission of the URC Service Planner is processing and

presenting the result which comes from the execution engine. The typical presenting type is the narration with voice using by TTS.

```
- <service:Service rdf:ID="LightControlService">
    <service:presents rdf:resource="#LightControlServiceProfile" />
    <service:describedBy rdf:resource="#LightControlServiceProcessModel" />
    <service:supports rdf:resource="#LightControlServiceGrounding" />
  </service:Service>
- <profile:Profile rdf:ID="LightControlServiceProfile">
    <service:presentedBy rdf:resource="#LightControlService" />
    <profile:has_process rdf:resource="#LightControlServiceProcessModel" />
    <profile:serviceName>LightControlService</profile:serviceName>
    <rdfs:label>LightControlService</rdfs:label>
    <profile:textDescription>Control light on the Demo-Room</profile:textDescription>
    <profile:hasInput rdf:resource="#Structure" />
    <profile:hasEffect rdf:resource="http://urcsp.etri.re.kr/concepts/2004/10/Behavior#Toggle/" />
  </profile:Profile>
- <process:ProcessModel rdf:ID="LightControlServiceProcessModel">
    <service:describes rdf:resource="#LightControlService" />
  - <process:hasProcess>
    - <process:AtomicProcess rdf:ID="LightControlServiceProcess">
      - <process:hasInput>
        - <process:Input rdf:ID="Structure">
            <process:parameterType
              rdf:resource="http://urcsp.etri.re.kr/concepts/2004/10/HouseStructure#Structure" />
          </process:Input>
        </process:hasInput>
        <process:hasEffect rdf:resource="http://urcsp.etri.re.kr/concepts/2004/10/Behavior#Toggle" />
      </process:AtomicProcess>
    </process:hasProcess>
  </process:ProcessModel>
- <grounding:WsdlGrounding rdf:ID="LightControlServiceGrounding">
    <service:supportedBy rdf:resource="#LightControlService" />
  - <grounding:hasAtomicProcessGrounding>
    - <grounding:WsdlAtomicProcessGrounding rdf:ID="ToggleServiceGrounding">
        <grounding:owlsProcess rdf:resource="#LightControlServiceProcess" />
        <grounding:wsdlDocument>http://129.254.164.143:8080/axis/services/LightControlServiceService?
          wsdl</grounding:wsdlDocument>
      - <grounding:wsdlOperation>
        - <grounding:WsdlOperationRef>
            <grounding:portType>LightControlServicePort</grounding:portType>
            <grounding:operation>toggleLight</grounding:operation>
          </grounding:WsdlOperationRef>
        </grounding:wsdlOperation>
      - <grounding:wsdlInputMessageParts rdf:parseType="Collection">
        - <grounding:wsdlMessageMap>
            <grounding:owlsParameter rdf:resource="#Structure" />
            <grounding:wsdlMessagePart>structure</grounding:wsdlMessagePart>
          </grounding:wsdlMessageMap>
        </grounding:wsdlInputMessageParts>
        <grounding:wsdlOutputMessage>toggleLightResponse</grounding:wsdlOutputMessage>
        <grounding:wsdlInputMessage>toggleLightRequest</grounding:wsdlInputMessage>
      </grounding:WsdlAtomicProcessGrounding>
    </grounding:hasAtomicProcessGrounding>
  </grounding:WsdlGrounding>
```

Fig. 2 Semantic Description for the Web service of the Light Control Service

4). Service Discovery

URC service discovery engine is a key component for the Intelligent Web Service Framework. Automatic Web services discovery involves automatically locating Web services that provide a particular service and that adhere to requested properties. With the semantic description in OWL-S of Web services, URC service discovery engine locates the requested Web service on the basis of the capabilities described in IOPE (Input, Output, Precondition, and Effect). In case those results are several Web services, ranking algorithm sorts results.

To interact with URC, URC service discovery engine use OWL-QL) [7] (Web Ontology Language - Query Language) for the query language and SOAP for the communication protocol. OWL-QL is a candidate standard language and protocol for query

answering dialogs among Semantic Web computational agents using knowledge represented in W3C's OWL. An OWL-QL query accompanies a KIF [8] formatted query which is used in a matching engine. URC query is converted to KIF query.

When the matching engine receives a query from a URC or another software agent, it searches its storage for fulfilling the incoming requests. A semantic markup of a certain Web services matches a request from the URC, the matching engine returns that services for the request.

The query processor extends a request query to create hierarchical ontology for the data type of the query. This process is required to find more services whose data types don't exactly match those of the requested query, but whose data types are ancestors/descendents of those in the requested query. Second, the query processor transforms the requested KIF query to the corresponding SQL query. The registry manager searches the storage to get the semantic markup for the appropriate Web service. The ranker estimates similarities between each Web services in search results and the extended hierarchical ontology, and ranks each Web service in the result set.

5). Services Execution

Service Execution system for URC includes four major components: Process Generator, Process Deployer, Process Executor, and Message Server. The Process Generator gets a service plan with IOPE from Service Planner, then searches Process Repository whether Process Repository already contains the same service plan or not. If Process Repository already has the service plan which we want, we get the process model from Service Repository directly instead of generating process model. In Process Repository, service plan is represented as a semantic graph using IOPE. If Process Repository does not contain service plan, then Process Generator produce process model.

The Process Deployer gets WSDLs and process model described as BPEL4WS from Process Generator. Process Deployer dispatches generated process model and makes client stub classes to invoke Web Service which is provided by Web service providers. To dispatch process model, first, Process Deployer makes client stub classes from WSDLs which describe the partner's operations and then convert process model to run-time process model which would be executed in Process Virtual Machine. Process run-time model includes input/output variables and process flow control information.

The Process manager executes run-time process model and manages process life-cycle. Process manager monitors process execution and conversation among partners. It also supports multi process management and long running process control. Long running process control is required in case of asynchronous invocation is used in run-time process model. Multi process management means URC Execution System is used as the server of many URC clients. Process Manager also manages persistent objects such as process instances, variables and process log. To response to URC Service Driver, Process Manager must get the XSL which describes context translation.

The Message Server manages every input message as response of Web Service invocation. It should know which partner deliver the input message. In addition, it should pipeline received message to exact process instance which is waiting some message delivery. Message server has Message Binder for the message transfer to Process Manager. Message Binder links all input ports to deployed process table which is containing correlation information of process model. Using Message Binder and deployed process table, input message precisely can be delivered to a process instance which is running in Process Virtual Machine.

4. EXPERIMENTS

To demonstrate our Intelligent Web Service Framework, we utilize two applications: One is the weather application from **the KMA (Korea Meteorological Administration)** which provides the weather information as the Web services. The other is the traffic application from **the KHC (Korea Highway Corporation)** which proffers traffic conditions of freeways in South Korea. With these two application, we simulates digital home services with the Light Control Web service which is connected to IR remote controller and is able to turn on/off the lamp in the laboratory. This Light Control Web service is by OWL-S, and is also registered in the URC Service Discovery Engine. The prototype of the URC equipped above components is shown in Figure 3.

URC includes following components.

- Navigation Module
- Voice Recognition Module
- Face Recognition Module
- Wireless Network Communication
- Robotic Control: through Web Services

We demonstrate URC services scenario in the laboratory home test-bed. The laboratory home equipped several electronic appliances such as a remote controlled lamp and a network-controlled blind. These appliance as well as URC are controlled by Web services, and are described by semantic markup in OWL-S.

Our test scenario is as follows.

1) A user asks URC to wake him up one hour later if it is rainy.
   - URC find a weather service through URC services registry and call a weather web service through the KMA web service.
   - URC wake up the user if it is rainy after one hour.

2) The user asks URC how long does it takes to arrive Seoul through the freeway.

- URC find the freeway information service through URC services registry and get information about the freeway through the KHC web service.

3) The user asks URC to turn off the lamp when the room is empty.
- URC find light control service through URC registry and call that service to turn off the lamp after the user goes out.

Fig. 3 The Ubiquitous Robotic Companion

5. CONCLUSION

Currently, many researches related on Semantic Web, Web Services, and Ubiquitous Web are ongoing by a lot of organizations. In this point, we have implemented a system for robot enabled the Semantic Web Services. We also tested the Web Services which control IR based electronic appliances such as a lamp, a TV, and a refrigerator. As a consequence, URC is able to control these appliances by automatically discovering and invoking pre-described Semantic Web services.

Consequently, we propose Internet Service Framework which provides the URC or other robots with the mechanism to access the WWW and to control many devices by Semantic Web Services. With this work, it is realized that robotic applications can be leveled up to new services and we found the confidence of the possibility and the effectiveness on the Semantic Web and the Web Services. And also we have got lessons that various environments exist such as embedded machine, PDA, and cellular phone, and should deeply consider those implementation Frameworks for the Ubiquitous Web.

Nevertheless, there are still a number of issues concerning on Semantic Web Services and the ubiquitous computing world, if we consider many types of devices which usually exist in embedded environments. These properties will lead to a lot of problems such as user interaction, variety of computing power, limitation of input or output device, user context and a new standard. Most of issues should be treated carefully from all parties such as industry and academia from now on.

ACKNOWLEDGEMENTS

This research is operated by “the project of URC (Ubiquitous Robotics Companion) technology development” as a national project by the Ministry of Information and Communication Republic of Korea.

REFERENCES

[1] S. A. McIlraith, T. C. Son, H. Zeng, “The Semantic Web Services,” IEEE Intelligent Systems, Vol. 16, Issue 2, IEEE, 2001, pp. 46-53.
[2] S. R. Oh, “IT based Intelligent Web Service Robot,” RCV03 Invited talk, Las Vegas, Oct.26.2003.
[3] Smith, Welty et al., “OWL Web Ontology Language Guide”, W3C Recommendation 10 Februry 2004.
[4] F. Buijze, P.F.A.d. Dijcker, A.A. Hornickel, “Universal Description, Discovery, and Integration (UDDI) 3.0.1,” 14 October 2003.
[5] David Martin at el., “OWL-based Web Service Ontology ver. 1.1,” http://www.daml.org/services/owl-s/
[6] IBM, "BPEL4WS: Business Process Execution Language for Web Services Version 1.1", http://www-106.ibm.com/developerworks/webservices/library/wsbpel/
[7] Richard Fikes, Pat Hayes, Ian Horrocks, “OWL-QL: A Language for Deductive Query Answering on the Semantic Web,” KSL Technical Report 03-14.
[8] Michael R. Genesereth, “Knowledge Interchange Format”, http://logic.stanford.edu/kif/kif.html