Towards a Ubiquitous Robotic Companion: Design and Implementation of Ubiquitous Robotic Service Framework

Young-Guk Ha, Joo-Chan Sohn, Young-Jo Cho, and Hyunsoo Yoon

In recent years, motivated by the emergence of ubiquitous computing technologies, a new class of networked robots, ubiquitous robots, has been introduced. The Ubiquitous Robotic Companion (URC) is our conceptual vision of ubiquitous service robots that provide users with the services they need, anytime and anywhere in ubiquitous computing environments. To realize the vision of URC, one of the essential requirements for robotic systems is to support ubiquity of services: that is, a robot service must be always available even though there are changes in the service environments. Specifically robotic systems need to be automatically interoperable with sensors and devices in current service environments, rather than statically preprogrammed for them. In this paper, the design and implementation of a semantic-based ubiquitous robotic space (SemanticURS) is presented. SemanticURS enables automated integration of networked robots into ubiquitous computing environments exploiting Semantic Web Services and AI-based planning technologies.

Keywords: Ubiquitous robotics, ubiquitous computing, networked robotics, service robotics, service planning, Semantic Web Services.

Manuscript received July 20, 2005; revised Sept. 15, 2005.

This work is financially supported by URC Technology R&D Program of Korea Ministry of Information and Communication.

Young-Guk Ha (phone: +82 42 860 6375, email: ygha@etri.re.kr), Joo-Chan Sohn (email: jcsohn@etri.re.kr), and Young-Jo Cho (email: youngjo@etri.re.kr) are with Intelligent Robot Research Division, ETRI, Daejeon, Korea.

Hyunsoo Yoon (email: hyoon@camars.kaist.ac.kr) is with the Department of Computer Science, KAIST, Korea.

I. Introduction

Due to the progress of communication network technologies, many people have been researching Internet-based networked robotic systems, which are mainly focused on tele-operation and monitoring of networked robotic devices and sensors (for example, mobile robots, unmanned vehicles, position sensors, and so on.) by human supervisors in Internet environments. Leveraging the advantages of Internet technology, such systems allow users from all over the world to visit museums, tend gardens, navigate undersea, or float in blimps 24 hours a day. They have great potential for industry, education, entertainment, and security by making valuable robotic hardware accessible to a broad audience.

Several attempts have been made to develop such Internet-based networked robotic and monitoring systems using the World Wide Web and distributed object technologies. The typical Web-based networked robotics approach uses HTTP combined with CGI (common gateway interface) or Java to control remote sensors and actuators: for instance, University of California's tele-excavation system, Mercury [1]; Carnegie Mellon University's indoor mobile robot, Xavier [2]; Ecole Polytechnique Fédérale de Lausanne's maze robot, KhepOnTheWeb [3]; Roger Williams University's PumaPaint [4]; and Pohang University of Science and Technology's XNMS [5]. Another approach for networked robotic systems is based on distributed object technology such as CORBA (common object request broker architecture) and Java RMI (remote method invocation): for instance, in NRSP (network robot service platform) [6] and DAIR (distributed architecture for Internet robot) [7].

In recent years, motivated by the emergence of ubiquitous computing [8] technologies as the next generation computing paradigm, a new class of networked robots, ubiquitous robots, has been introduced [9], [10]. Actually, they are networked robots integrated into ubiquitous computing environments including networked sensors and actuators. They can even be realized as a form of ubiquitous computing environments themselves such as in a robotic room [11]. The Ubiquitous Robotic Companion (URC) [12] is our conceptual vision of a ubiquitous service robot that provides users with the services they need, anytime, anywhere in ubiquitous computing environments. To realize the vision of URC, one of the essential requirements for the robotic systems is to support a ubiquity of services: that is, a robot service must always be available even though there are changes in the service environments. The current networked robotics approaches are, as mentioned above, mainly focused on a behavior-oriented tele-operation of remote robotic devices with Web applications or distributed objects programmed for specific environments. Surely, they can help people with overcoming the limit of time and location for services. To provide ubiquitous services, however, robotic systems need to be automatically interoperable with ubiquitous sensors and devices in the current service environments, rather than statically preprogrammed for them.

In this paper, a semantic-based ubiquitous robotic space (SemanticURS) framework is presented. It enables automated integration of networked robots into ubiquitous computing environments in a service-oriented way. SemanticURS exploits Semantic Web Services [13], a state of the art Web technology, and an AI-based planning technique to provide automated interoperation between networked robots and ubiquitous computing devices in service environments. That is, Web Services [14] for robots, networked sensors, and devices are implemented as a unified interface method for accessing them. Then, knowledge about such Web Services is described in Web ontology language for services (OWL-S) [15], the semantic description language for Web Services, and registered to environmental knowledge bases (KBs), so that a robotic agent can automatically discover the required knowledge and compose a feasible service plan for the current environments. Next, the agent provides service by automatically interacting with robots, sensors, and devices through the simple object access protocol (SOAP) [16], the Web Services execution protocol, according to the service plan, as shown in a comparison in Figs. 1(a) and 1(b).

The rest of the paper is organized as follows. Section II briefly introduces the Semantic Web Services technologies as fundamental background. Section III describes the detailed design of SemanticURS, and section IV explains the prototype implementation and experiments in our networked home test bed. Finally, section V gives conclusions and future works.

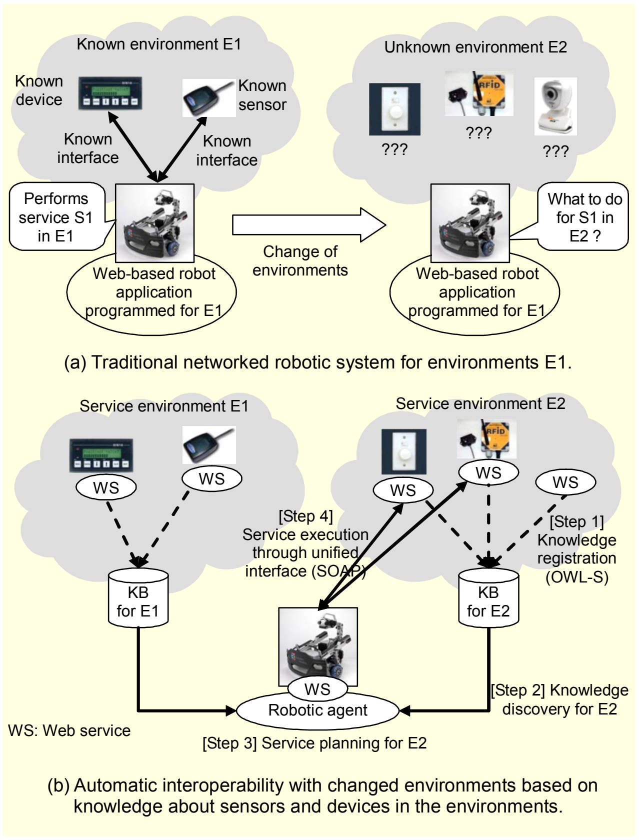

Known environment E1
Known device
Known sensor
Known interface
Known interface
Performs service S1 in E1
Web-based robot application programmed for E1
Change of environments
Unknown environment E2
???
???
???
What to do for S1 in E2 ?
Web-based robot application programmed for E1
(a) Traditional networked robotic system for environments E1.
Service environment E1
WS
WS
KB for E1
Service environment E2
WS
WS
WS
[Step 4] Service execution through unified interface (SOAP)
[Step 1] Knowledge registration (OWL-S)
KB for E2
WS
Robotic agent
[Step 2] Knowledge discovery for E2
[Step 3] Service planning for E2
WS: Web service
(b) Automatic interoperability with changed environments based on knowledge about sensors and devices in the environments.

Fig. 1. Comparison with a traditional networked robot system.

II. Semantic Web Services

The Web, once a repository of text and images, is evolving into a provider of services: information-providing services, such as Internet information providers and portals; and world-altering services, such as e-commerce and e-business applications. Web-accessible programs and databases realize these services through CGI, Java, ActiveX, or the Web Services [14] technology. Fundamental to having computer programs or agents implement reliable and automated interoperation of such services is the need to make the services computer interpretable—to create a semantic Web [17] of services whose semantics, such as properties, capabilities, and interfaces are encoded in an unambiguous, machine-understandable form. The Semantic Web Services technology is developed to meet this need by describing Web Services with an OWL [18] ontology, namely OWL-S [15], which provides AI-inspired markups for specifying a richer-level of service semantics. As shown in Fig. 2, OWL-S markups are grouped into three essential classes.

Service Profile: This tells "what the service does"; that is, it gives the types of information needed by a service requester agent to determine whether the service meets its needs. In addition to representing the capabilities of a service, the profile can be used to express the needs of the service requester agent so

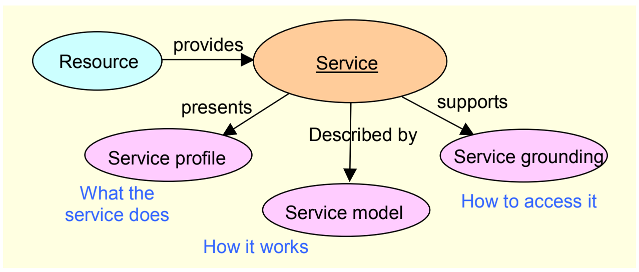


Fig. 2. Top level of the OWL-S ontology.

that a matchmaker has a convenient dual-purpose representation upon which to base its operations. The OWL-S profile ontology provides the following markups to describe properties of a service profile: Profile, serviceName, textDescription, hasInput, hasOutput, hasPrecondition, hasEffect, serviceCategory, and so on.

Service Model: This tells "how the service works"; that is, it describes what happens when an atomic or composite service is carried out. This description may often be used by a service requester agent in the following ways: to perform a more in-depth analysis of whether the service meets its needs; to compose service descriptions from multiple services to achieve a specific goal; during the course of the service enactment, to coordinate the activities of the different participants; and to monitor the execution of the service. The OWL-S process ontology provides the following markups to model services as processes: ProcessModel, AtomicProcess, CompositeProcess, SimpleProcess, ProcessComponent, ControlConstruct, Sequence, Choice, Repeat-Until, and so on.

Service Grounding: This specifies the details of how an agent can access a service. Typically, a grounding will specify a communication protocol, message formats, and other service-specific details such as port numbers used in contacting the service. In addition, the grounding must specify, for each abstract type specified in the service model, an unambiguous way of exchanging data elements of that type with the service. The OWL-S grounding ontology provides the following markups to ground OWL-S atomic services with concrete Web Services whose interfaces are described in the Web Services Description Language (WSDL) [19]: WsdlGrounding, hasAtomicProcessGrounding, WsdlAtomicProcessGrounding, wsdlOperation, wsdlDocument, WsdlOperationRef, portType operation, and so on.

III. Design of SemanticURS

1. The Architecture of SemanticURS

As shown in Fig. 3, the architecture of SemanticURS consists of three major components, a robotic agent (RA), device web services (DWS), and an environmental knowledge

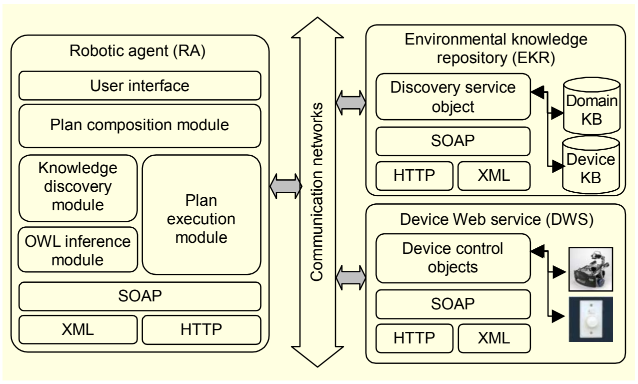
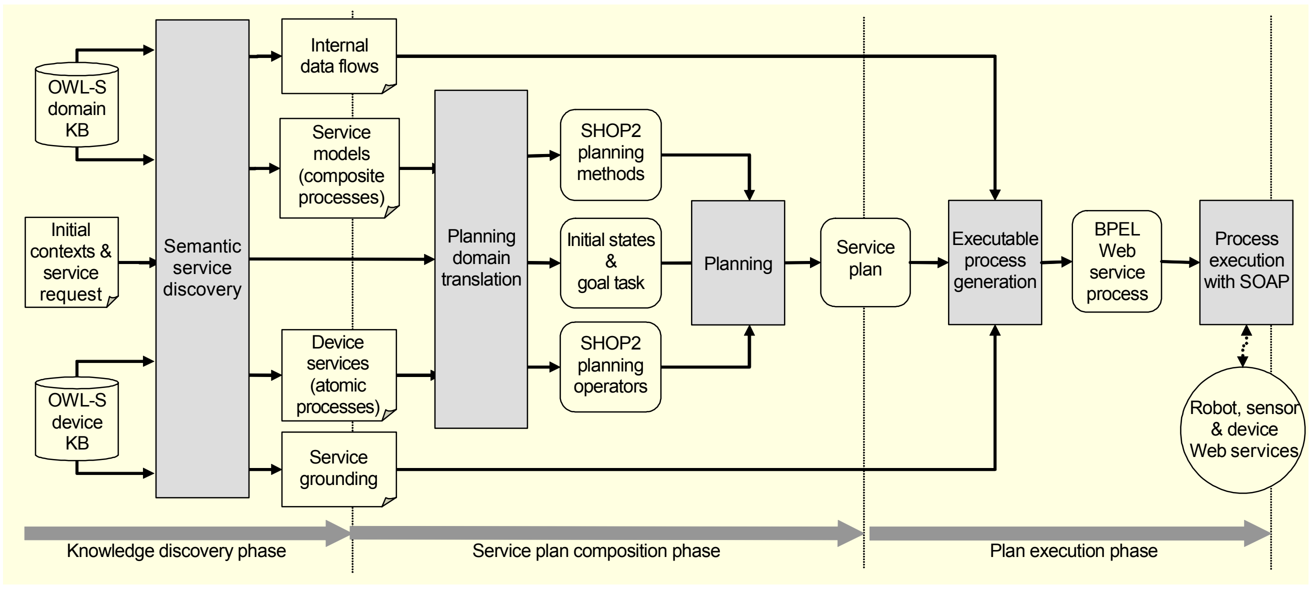


Fig. 3. Detailed architecture of SemanticURS.

repository (EKR).

The RA, as an intelligent planning and service requester agent, plays the major role of an automated integration procedure in the SemanticURS architecture. The RA consists of a user interface, a plan composition module, a knowledge discovery module, an OWL inference module, a plan execution module, and a communication stack for Web Services execution including SOAP [16], XML and HTTP. A user can input a service request with the user interface, and optionally initial service contexts can be inputs for the service. Then, the service request (and optional service contexts) is encoded with vocabularies in OWL-S process ontology, so that the plan composition module can understand a user's service request and automatically discover the required knowledge for planning through the knowledge discovery module. To search KBs in the EKR for the required knowledge, the knowledge discovery module creates semantic discovery queries encoded in the resource description framework (RDF) data query language (RDQL) [20], [21] by reasoning about the semantics of the service request with the OWL inference module. After the composition of a service plan, the plan execution module translates the service plan into an executable format and executes it through the Web Services communication stack.

The EKR contains a domain KB and a device KB, which are used for environmental knowledge registration and discovery. The domain KB stores OWL-S knowledge of general service concepts and composite tasks with internal data flows describing common service models for a certain service domain. The device KB stores OWL-S knowledge of atomic tasks representing device or sensor services in specific service environments and corresponding service groundings for them. And the EKR includes the discovery service object to handle knowledge discovery queries with semantic predicates on OWL-S knowledge from the RA. The EKR also includes a Web Services communication stack because it works as a Web Service itself, that is, the RA can access the EKR with SOAP, the unified interface protocol in our framework.



OWL-S domain KB
Initial contexts & service request
OWL-S device KB
Semantic service discovery
Internal data flows
Service models (composite processes)
Device services (atomic processes)
Service grounding
Planning domain translation
SHOP2 planning methods
Initial states & goal task
SHOP2 planning operators
Planning
Service plan
Executable process generation
BPEL Web service process
Process execution with SOAP
Robot, sensor & device Web services
Knowledge discovery phase
Service plan composition phase
Plan execution phase

Fig. 4. Automated integration procedure of the SemanticURS agent.

The DWS is an implementation of Web Services for ubiquitous devices including robots, sensors, actuators, and so on. Each DWS can have control objects for one device or multiple devices, which may work cooperatively: for instance, an air-conditioning device and a temperature sensor. And the DWS also has a Web Services communication stack to communicate with the RA.

Figure 4 shows the overall automated integration procedure of the SemanticURS agent, which consists of knowledge discovery, service plan composition, and plan execution phases. The following subsections will explain each phase in more detail.

2. Knowledge Description and Discovery

As mentioned above, knowledge about common service models and device services needs to be described in OWL-S ontology so that the RA can discover the required knowledge to compose a service plan for a user request. Knowledge about device services is described as OWL-S atomic processes [15] and used to generate the space of feasible actions (primitive tasks) for plan composition in specific service environments. Knowledge about common service models is described as OWL-S composite processes [15] and used to constrain how the service plans are to be composed independently of specific service environments. Some running examples of the knowledge descriptions will be presented in the experiments section.

The knowledge discovery procedure of the RA is performed by sending discovery queries to the EKR and receiving corresponding responses. This procedure is initiated by a service request input from the user. Once a service is requested, it is encoded with OWL-S process ontology. That is, the service request is translated into an OWL-S simple process [15] describing the user-requested service. Then, a service model for the user request and required device services are found from the domain KB and the device KB using a discovery algorithm. In the knowledge discovery module of the RA, we implement the semantic service discovery algorithm, which finds an extended set of device services to compose a feasible plan for the requested service. That is, the set of device services acquired during the knowledge discovery phase can be extended by reasoning about the semantics of a service model for the user request. Consequently, the semantic service discovery algorithm finds a compatible service (a semantically replaceable service) for every device service that is not discovered by exact query matching.

As formerly explained, knowledge about service models and device services are described as OWL-S composite and atomic processes, which have OWL individuals [18] as ranges of their input, output, precondition, and effect (IOPE) property [15] values. Therefore, we will define some semantic relations between OWL individuals and OWL-S processes in predicate logic syntax to describe the algorithm formally: that is, p(s, o), where p is a predicate, s is a subject, and o is an object. Every argument is an OWL individual if not specifically defined. And the semantics of each predicate or argument is based on the formal semantics of RDF [20], [22], RDFS (RDF schema) [22], [23], OWL (Full) [18], [24] and OWL-S process ontology [15], [25] as it is prefixed.

Definition. Subsumption relation, $x \supseteq y$.

An OWL individual x of Class a subsumes an OWL individual y of Class b if and only if rdfs:subClassOf(b, a) $\vee$ owl:equivalentClass(a, b) holds.

Definition. Subsumption Relation on a Property, $x \supseteq y / p$.

An OWL individual x subsumes an OWL individual y on an OWL property p if and only if for each $a \in \{v \mid p(x, v)\}$, there exists a distinct $b \in \{u \mid p(y, u)\}$ s.t. $a \supseteq b$.

Definition. Compatibility Relation, $x \sim y$.

An OWL-S process x is compatible with an OWL-S process y if and only if $(x \supseteq y$ / process:hasInput$) \wedge (x \supseteq y$ / process:hasPrecondition$) \wedge (y \supseteq x$ / process:hasOutput$) \wedge (y \supseteq x$ / process:hasEffect$)$.

Definition. Equivalence Relation, $x \equiv y$.

An OWL-S process x is equivalent to an OWL-S process y if and only if x and y have the same set of IOPE property values.

Algorithm. Semantic_Service_Discovery(R, D).

```
Input: A service request R;
Output: A set of device services D;
Procedure:
  Discover a service model M from the domain KB
      s.t. $M \equiv R$;
  If (the discovery fails) exit with Discovery_Error;

  Set D = $\emptyset$;
  For each component process P in M {
      If (P is a composite process)
          call Semantic_Service_Discovery(P, D);

      Discover a device service S from the device KB
          s.t. atomic process of S $\equiv$ P;
      If (the discovery succeeds) Add S to D;
      else {
          Discover a device service S from the device KB
            s.t. atomic process of S $\sim$ P;
          If (the discovery succeeds) {
              Add S to D;
              Replace P in the service model M with
                  atomic process of S;
          } else exit with Discovery_Error;
      }
  }
```

3. Service Plan Composition

In the plan composition phase, the SemanticURS agent automatically composes service plans for the user request using hierarchical task network (HTN) planning [26]. HTN planning is an AI planning methodology that creates plans by a task decomposition process in which the planner decomposes tasks into smaller subtasks until primitive tasks, which can be performed directly, are found. The entire task decomposition process is based on planning operators and methods called a planning domain. Such task decomposition concept and modularity of HTN planning is very similar to the concept of composite and atomic processes in OWL-S. In addition, HTN planning supports expressive domain and precondition representation to solve the complex planning problems efficiently. To perform HTN planning, it is required to translate OWL-S knowledge found in the discovery phase into the HTN planning domain: that is, translate service model knowledge into the planning methods and device service knowledge into the planning operators.

We programmed such translation algorithm into the plan composition module of the RA. In the SemanticURS framework, we use a domain-independent HTN planning system, SHOP2 [27]. Therefore, generating a planning domain in terms of SHOP2 domain is required. Inheriting a generic HTN domain, each SHOP2 operator describes what needs to be done to accomplish some primitive task, and each SHOP2 method tells how to decompose some compound task into partially ordered subtasks. A SHOP2 operator is an expression of the form (p(v) Pre Del Add), where p(v) is a primitive task with a list of input parameters v, Pre represents the operator's preconditions, Del represents the operator's delete list that includes a list of things that will become false after an operator's execution, and Add represents the operator's add list that includes a list of things that will become true after the operator's execution. Knowledge about a device service, described as an OWL-S atomic process A, is translated into a SHOP2 planning operator by replacing v with a set of inputs of A, Pre with conjunction of all the preconditions of A, and Add with conjunction of all the effects of A. The SHOP2 method is an expression of the form ($c(v)$ Pre_1 T_1 Pre_2 T_2 …), where c(v) is a compound task with a list of input parameters v, each Pre_i is a precondition expression, and each T_i is a partially ordered set of subtasks. Knowledge about a service model described as an OWL-S composite process C is translated into a set of SHOP2 planning methods according to the control constructs of C. For example, in the case of a sequence control construct, v is replaced with a set of inputs of C, Pre_i with conjunction of all the preconditions of a component process P_i, and T_i with P_i. For the specific SHOP2 domain generation algorithm, refer to [27].

4. Service Plan Execution

The result of the plan composition phase is a sequence of primitive tasks, that is, a sequence of OWL-S atomic processes

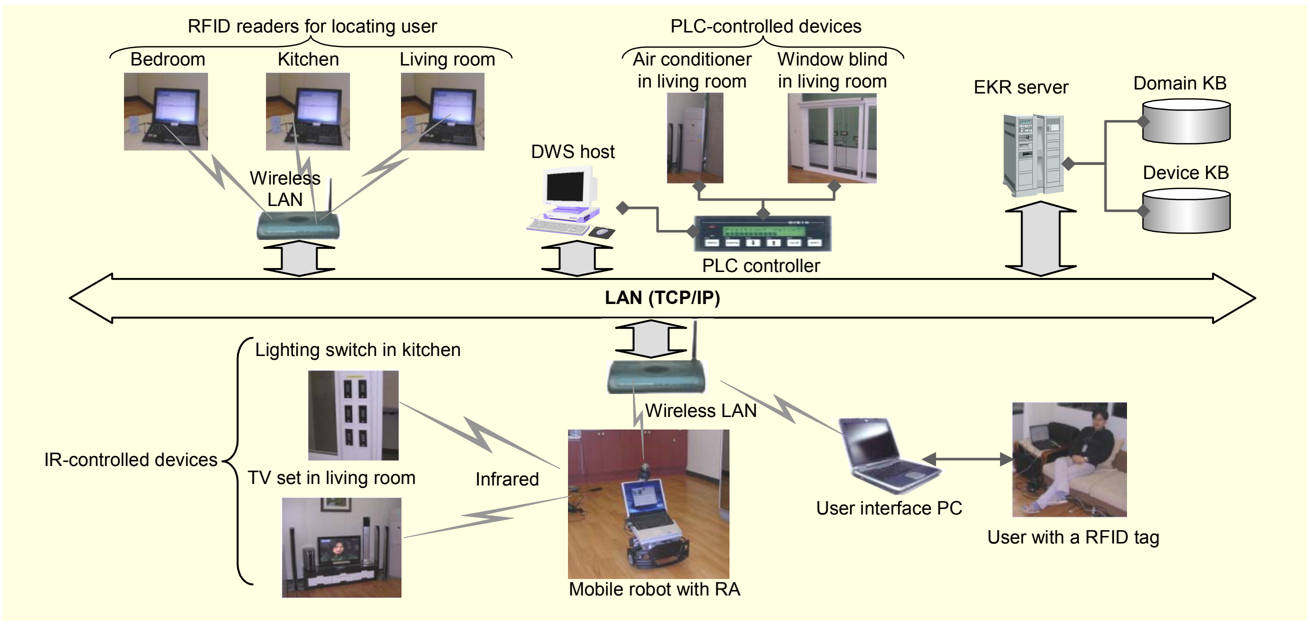


Fig. 5. Experimental environments: mobile robot in networked home test bed.

with corresponding service grounding. To be executed, a service plan must be translated into an executable format such as a process of concrete Web Services. In our implementation, BPEL4WS (business process execution language for Web services) [28] is used to create executable Web Service processes at the process execution phase. As shown in Fig. 4, knowledge about data flows in service models and service grounding is used to generate an executable service process from the plan composition result. We use IBM's BPWS4J (BPEL4WS for Java) platform to implement BPEL process generation and execution functionalities of the plan execution module. At this time, we do not consider a monitoring and contingency planning feature for the system during the execution of a service process. However, it is important for system robustness and will be one of our next research and development issues.

IV. Experiments

1. Experimental System

We implement a SemanticURS prototype system and make experiments in our networked home test bed, which has a bedroom, kitchen, and living room. As shown in Fig. 5, the experimental environments include an ERSP Scorpion robot with vSLAM navigation [29]. It is equipped with a USB-connected infrared (IR) remote controller. The environments also include IR-controlled devices such as a TV set and lighting switches, a power line communication (PLC) controller, and PLC-controlled devices such as an air conditioner and a

Table 1. Web services implementations.

Name	Description	Device	Location
getUser Location	Returns which RFID reader senses given user ID in its radio boundary	RFID readers	DWS host
raiseWindow Blind	Raises window blind with PLC controller	PLC controller	DWS host
pullDown WindowBlind	Pulls down window blind with PLC controller	PLC controller	DWS host
turnOnAir Conditioner	Turns on air conditioner with PLC controller	PLC controller	DWS host
turnOffAir Conditioner	Turns off air conditioner with PLC controller	PLC controller	DWS host
turnOnLight	Turns on lighting with IR remote controller	IR remote controller	PC on the robot
turnOffLight	Turns off lighting with IR remote controller	IR remote controller	PC on the robot
turnOnTV	Turns on TV set with IR remote controller	IR remote controller	PC on the robot
turnOffTV	Turns off TV set with IR remote controller	IR remote controller	PC on the robot
moveTo	Moves robot to the specific point of the given place	Mobile robot	PC on the robot

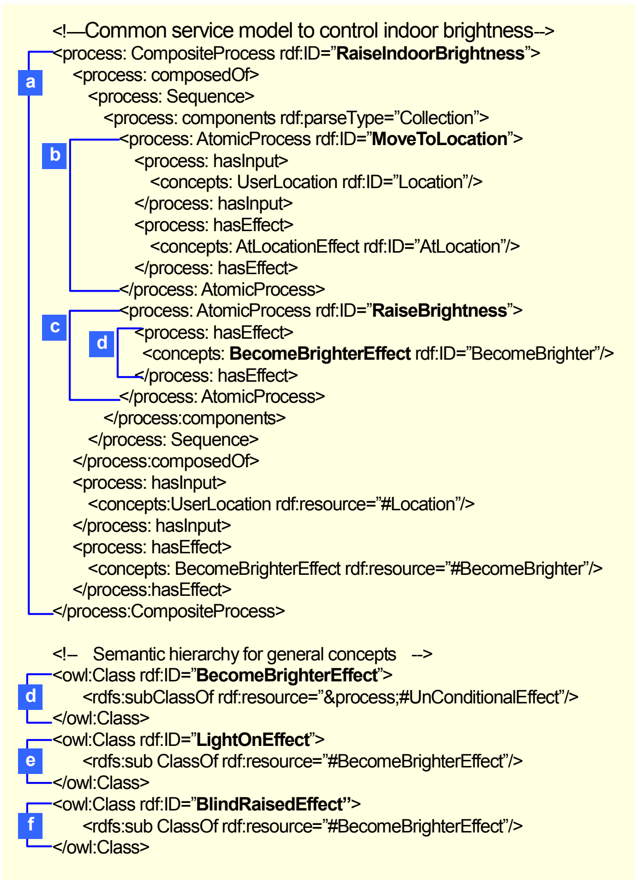
motorized window blind. In addition, there are three radio frequency identification (RFID) readers in each place to sense the location of the user who carries a RFID tag. The SemanticURS prototype implementation consists of a DWS Host, EKR Server, laptop PC for RA mounted on the ERSP

Scorpion robot, and an additional laptop PC for Web-based user interfaces. They are connected to each other via a wired or wireless LAN.

We use Java2 SDK as the development platform for the SemanticURS prototype system and HP's Jena Semantic Web framework to implement OWL ontology reasoning functionality for RA and knowledge bases for EKR. We also use Apache AXIS Web Services toolkit to implement Web Services communication features for RA, DWS and EKR. Table 1 lists concrete Web Services that are implemented for sensors and devices in our networked home test bed including RFID readers, a Scorpion mobile robot, a PLC controller, and an IR remote controller.

2. Knowledge Bases

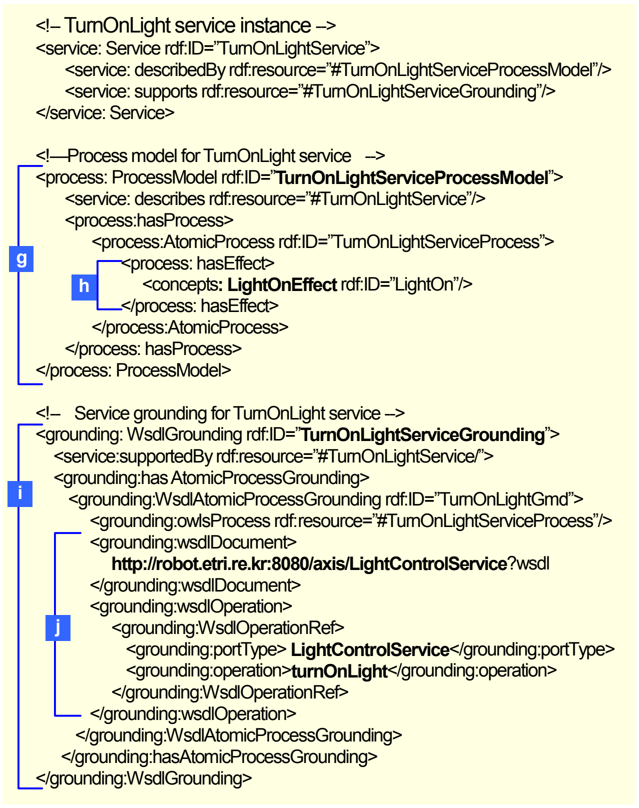
The experimental domain KB contains OWL-S service models and general concepts to describe the semantics of robotic services for the networked home environments shown in Fig. 5. In the domain KB, the service concepts construct a subsumption hierarchy. They provide semantic values for inputs, outputs, preconditions, and effects properties of service models and device services in the experimental environments.



```
<!—Common service model to control indoor brightness–>
<process: CompositeProcess rdf:ID="RaiseIndoorBrightness">
  <process: composedOf>
    <process: Sequence>
      <process: components rdf:parseType="Collection">
        <process: AtomicProcess rdf:ID="MoveToLocation">
          <process: hasInput>
            <concepts: UserLocation rdf:ID="Location"/>
          </process: hasInput>
          <process: hasEffect>
            <concepts: AtLocationEffect rdf:ID="AtLocation"/>
          </process: hasEffect>
        </process: AtomicProcess>
        <process: AtomicProcess rdf:ID="RaiseBrightness">
          <process: hasEffect>
            <concepts: BecomeBrighterEffect rdf:ID="BecomeBrighter"/>
          </process: hasEffect>
        </process: AtomicProcess>
      </process:components>
    </process: Sequence>
  </process:composedOf>
  <process: hasInput>
    <concepts:UserLocation rdf:resource="#Location"/>
  </process: hasInput>
  <process: hasEffect>
    <concepts: BecomeBrighterEffect rdf:resource="#BecomeBrighter"/>
  </process:hasEffect>
</process:CompositeProcess>

<!–  Semantic hierarchy for general concepts  –>
<owl:Class rdf:ID="BecomeBrighterEffect">
    <rdfs:subClassOf rdf:resource="&process;#UnConditionalEffect"/>
</owl:Class>
<owl:Class rdf:ID="LightOnEffect">
    <rdfs:sub ClassOf rdf:resource="#BecomeBrighterEffect"/>
</owl:Class>
<owl:Class rdf:ID="BlindRaisedEffect">
    <rdfs:sub ClassOf rdf:resource="#BecomeBrighterEffect"/>
</owl:Class>
```

Fig. 6. OWL-S knowledge for a service model and its concepts.



```
<!– TurnOnLight service instance –>
<service: Service rdf:ID="TurnOnLightService">
    <service: describedBy rdf:resource="#TurnOnLightServiceProcessModel"/>
    <service: supports rdf:resource="#TurnOnLightServiceGrounding"/>
</service: Service>

<!—Process model for TurnOnLight service   –>
<process: ProcessModel rdf:ID="TurnOnLightServiceProcessModel">
    <service: describes rdf:resource="#TurnOnLightService"/>
    <process:hasProcess>
        <process:AtomicProcess rdf:ID="TurnOnLightServiceProcess">
            <process: hasEffect>
                <concepts: LightOnEffect rdf:ID="LightOn"/>
            </process: hasEffect>
        </process:AtomicProcess>
    </process: hasProcess>
</process: ProcessModel>

<!–  Service grounding for TurnOnLight service –>
<grounding: WsdlGrounding rdf:ID="TurnOnLightServiceGrounding">
  <service:supportedBy rdf:resource="#TurnOnLightService/">
  <grounding:has AtomicProcessGrounding>
    <grounding:WsdlAtomicProcessGrounding rdf:ID="TurnOnLightGrnd">
      <grounding:owlsProcess rdf:resource="#TurnOnLightServiceProcess"/>
      <grounding:wsdlDocument>
        http://robot.etri.re.kr:8080/axis/LightControlService?wsdl
      </grounding:wsdlDocument>
      <grounding:wsdlOperation>
        <grounding:WsdlOperationRef>
          <grounding:portType> LightControlService</grounding:portType>
          <grounding:operation>turnOnLight</grounding:operation>
        </grounding:WsdlOperationRef>
      </grounding:wsdlOperation>
    </grounding:WsdlAtomicProcessGrounding>
  </grounding:hasAtomicProcessGrounding>
</grounding:WsdlGrounding>
```

Fig. 7. OWL-S knowledge for a lighting device service.

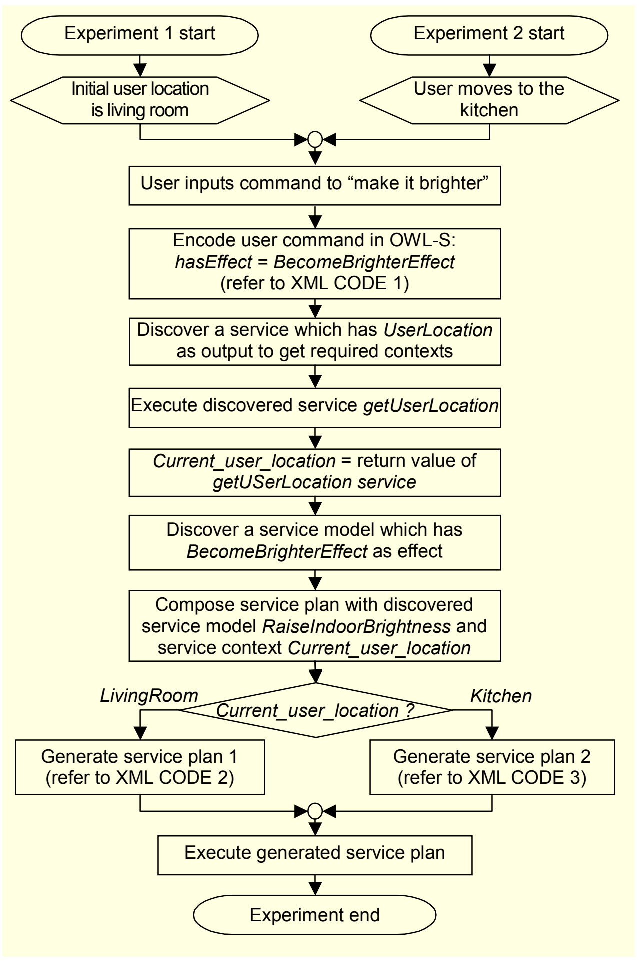
The example knowledge of Fig. 6 shows a portion of a common service model and a concept hierarchy for an experimental networked home service that controls indoor brightness. As shown in the figure, the RaiseIndoorBrightness service (part a) is composed of a sequence of two atomic processes, MoveToLocation (part b) and RaiseBrightness (part c). The first one is to navigate the robot to the given user's location, and the second one is to raise the brightness of the place. In more detail, the RaiseBrightness process has a BecomeBrighterEffect (part d) concept as its effect, which is a super-concept of LightOnEffect (part e) and BlindRaisedEffect (part f) as described in the highlighted codes of the Figure.

The device KB contains OWL-S knowledge about the implemented device Web Services listed in Table 1. The example of Fig. 7 shows knowledge for a turnOnLight service that consists of TurnOnLightServiceProcessModel (part g) and TurnOnLightServiceGrounding (part i). As described in the highlighted codes, the device service has a LightOnEffect (part h) concept as its effect, which is a sub-concept of the BecomeBrighterEffect (part d) concept. The service grounding connects the device service with the concrete turnOnLight service implementation in http://robot.etri.re.kr:8080/axis/LightControlService (part j), the Web Service object for a lighting control device existing in the experimental environments.

3. Experimental Results

In the beginning of the experiment, the user is sitting on a sofa in a living room carrying his RFID tag, and the robot is in the kitchen. The user commands the robot to "Make it brighter" through the Web-based user interface for the RA. As mentioned in the previous sections, the user command is described as an OWL-S simple process that has the BecomeBrighterEffect concept, part d in Fig. 6, as its effect, as shown in the following code in line (1).

```
<!—
      XML CODE 1 : OWL-S encoded user command
—>
<process:SimpleProcess rdf:ID="Cmd000">
  <process:hasEffect>
    <concepts:BecomeBrighterEffect rdf:ID="E000"/>     (1)
  </process:hasEffect>
</process:SimpleProcess>
```



Experiment 1 start
Experiment 2 start
Initial user location is living room
User moves to the kitchen
User inputs command to "make it brighter"
Encode user command in OWL-S: hasEffect = BecomeBrighterEffect (refer to XML CODE 1)
Discover a service which has UserLocation as output to get required contexts
Execute discovered service getUserLocation
Current_user_location = return value of getUSerLocation service
Discover a service model which has BecomeBrighterEffect as effect
Compose service plan with discovered service model RaiseIndoorBrightness and service context Current_user_location
LivingRoom
Current_user_location ?
Kitchen
Generate service plan 1 (refer to XML CODE 2)
Generate service plan 2 (refer to XML CODE 3)
Execute generated service plan
Experiment end

Fig. 8. Flowchart of the experimental procedures.

As illustrated in the procedure of the first experiment in Fig. 8, to perform the requested service, the RA must know the current location of the user as one of the essential contexts for services. Then, the RA sends the EKR server a discovery query for a service that has UserLocation as its output. As the result of querying, the RA receives OWL-S knowledge of the getUserLocation service and executes it based on its service grounding. In the experiment, the getUserLocation service responds to the agent with LivingRoom as the current location of the user by scanning the RFID readers.

After knowing the current location of the user, the RA sends the EKR server a query for a service model that has BecomeBrighterEffect as its effect. As the result of querying, the agent receives OWL-S knowledge of the indoor brightness control service model, RaiseIndoor Brightness, described in Fig. 6, and then performs a device service discovery and service plan composition with it. During the device service discovery phase, the RA discovers feasible device services by sending the EKR server extended queries as described in the semantic service discovery algorithm of section III.2. In the case of the "Make it brighter" command, the RA tries to find a device service that has BecomeBrighterEffect as its effect by sending a direct match query according to the service model. When it fails, the RA tries to find a device service that has a sub-concept of BecomeBrighterEffect as its effect, such as LightOnEffect or BlindRaisedEffect, by sending an extended query. In addition, the RA sends the EKR server the service context Current_user_location = LivingRoom with every query for context-aware service discovery. As a result, MoveToService in line (2), the OWL-S knowledge about moveTo service that has AtLocationEffect as its effect, and RaiseWindowBlindService in line (3), the OWL-S knowledge about raiseWindowBlind service that has BlindRaisedEffect as its effect, are discovered and the following plan is generated with XML.

```
<!—
      XML CODE 2 : Service plan for experiment 1
—>
<plan type="Sequence">
<service  name="MoveToService">                          (2)
      <input name="Location" value="LivingRoom"/>
  </service>
  <service  name="RaiseWindowBlindService"/>             (3)
</plan>
```

Finally, the plan was successfully translated into the BPEL4WS process and executed in the plan execution phase as shown in the upper left and right photos of Fig. 9.

The second experiment is the same as the first one, but the service context Current_user_location changed to Kitchen (refer to the procedure of the second experiment in Fig. 8). That is, from the robot's point of view, the service environments changed. Reactively to this change, the RA decides to move to the Kitchen in line (4) and discovers the appropriate device service TurnOnLightService in line (5) that has LightOnEffect (part e in Fig. 6), the sub-concept of BecomeBrighterEffect, as its effect and composes the following service plan automatically.

```
<!—
        XML CODE 3 : Service plan for experiment 2
—>
<plan type="Sequence">
   <service name="MoveToService">
      <input name="Location" value="Kitchen"/>             (4)
   </service>
   <service  name="TurnOnLightService"/>                   (5)
</plan>
```

As a result of the second plan execution, the Scorpion robot successfully moved to the kitchen and turned on the lighting with the IR remote controller, instead of raising the window blind in the living room, as shown in the lower left and right photos of Fig. 9.

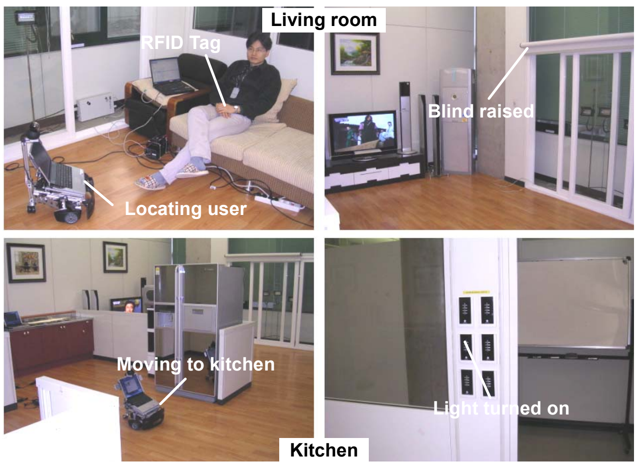


Fig. 9. Snapshots of the experiments.

V. Conclusions

In this paper, the SemanticURS framework is proposed and its prototype system for a networked home test bed implemented. SemanticURS enables automated integration of networked service robots into ubiquitous computing environments including wireless sensors and actuators to provide ubiquity of robotic services. SemanticURS exploits Semantic Web Services technology and an AI-based planning technique to support automated interoperation between service robots, wireless sensors, and service devices connected to each other in the ubiquitous networking environments. That is, Web Services for robots, sensors, and devices are implemented as the unified interface method for accessing them. Then, knowledge about the Web Services is described in OWL-S, the Semantic Web Services description language, and registered to the environmental knowledge bases so that a robotic agent can automatically discover the required knowledge and compose a feasible service plan for the current service environments. In addition, the proposed framework can also be applied to the development of intelligent software agents for a variety of service domains in ubiquitous computing environments.

Our future research direction will be focused on adapting SemanticURS to mobile ad hoc service environments. Currently, SemanticURS agents can discover required knowledge from centralized environmental knowledge bases, which are assumed to be well-known and always available to them. However, real ubiquitous computing environments will be mostly based on ad hoc networks that are spontaneous, completely distributed, and dynamic in nature. And they will surely consist of a huge number of mobile devices and sensors. This means that the centralized discovery approach is likely to suffer from a serious scalability problem in real ubiquitous computing environments. And it will be another challenge for the agents to maintain seamless connectivity to the centralized knowledge bases over mobile ad hoc networks. So, our future work will include how to discover and plan with the knowledge that is completely distributed to various ad hoc sensors and devices in the service environments. It will also include how to cooperate and share the knowledge between multiple robotic agents under the same or related service environments.

Another important future research focus is security and privacy issues in SemanticURS. We are planning to approach these issues basically from a role-based access control mechanism and XML digital signature technologies using cryptography and the public key infrastructure. In addition, we are also planning to use an ad hoc cluster-based security approach [30] for dynamic ad hoc service environments.

References

[1] K. Goldberg, S. Gentner, and C. Sutter et al., "The Mercury Project: A Feasibility Study for Internet Robotics," *IEEE Robotics and Automation Magazine*, vol. 7, no. 1, 2000, pp. 35-40.

[2] R. Simmons, "Xavier: An Autonomous Mobile Robot on the Web," *Proc. IEEE/RSJ Conf. on Intelligent Robots and Systems; Workshop on Web Robots*, Victoria, B.C. Canada, Oct. 1998, URL:

http://www.ri.cmu.edu/pub_files/pub1/simmons_reid_1999_1/simmons_reid_1999_1.pdf.

[3] P. Saucy and F. Mondada, "Open Access to a Mobile Robot on the Internet," *IEEE Robotics and Automation Magazine*, vol. 7, no. 1, 2000, pp. 41-47.

[4] M.R. Stein, "Interactive Internet Artistry," *IEEE Robotics and Automation Magazine*, vol. 7, no. 1, 2000, pp. 28-32.

[5] M. Choi, J. Hong, and H. Ju, "XML-Based Network Management for IP Networks," *ETRI J.*, vol. 25, no. 6, Dec. 2003, pp. 445-463.

[6] D. Wang, X. Ma, and X. Dai, "Web-Based Robotic Control System with Flexible Framework," *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA 2004)*, New Orleans, LA, Apr. 2004, pp. 3351-3356.

[7] X. Hou and J. Su, "A Distributed Architecture for Internet Robotics," *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA 2004)*, New Orleans, LA, Apr. 2004, pp. 3357-3362.

[8] M. Weiser, "Ubiquitous Computing," 1996, URL: http://www.ubiq.com/hyper text/weiser/UbiHome.html.

[9] J.H. Kim, Y.D. Kim, and K.H. Lee, "The Third Generation of Robotics: Ubiquitous Robot," *Proc. ICARA2004*, New Zealand, Dec. 2004, URL: http://www-ist.massey.ac.nz/conferences/icara2004/files/Papers/Paper01_ICARA2004_001_007.pdf. .

[10] K. Kiyoshi, "Ubiquitous Intelligent Robotics," *ATR UptoDate*, no. 5, 2003, URL: http://results.atr.jp/uptodate/ATR_2003 sum/kogure.html.

[11] T. Sato, T. Harada, and T. Mori, "Environment-Type Robot System 'Robotic Room' Featured by Behavior Media, Behavior Contents, and Behavior Adaptation," *IEEE/ASME Trans. on Mechatronics*, vol. 9, no. 3, Sept. 2004, pp. 529-534.

[12] S.R. Oh, "IT Based Intelligent Service Robot," *Proc. First NSF PI Workshop on Robotics and Computer Vision (RCV'03)* Invited talk, Las Vegas, Oct. 2003, URL: http://www.vcl.uh.edu/~rcv03/materials/slides/SangRok.ppt.

[13] S.A. McIlraith, T.C. Son, and H. Zeng, "The Semantic Web Services," *IEEE Intelligent Systems*, vol. 16, Issue 2, IEEE, 2001, pp. 46-53.

[14] W3C Recommendation, *Web Services Architecture*, W3C, 2004, URL: http://www.w3c.org/TR/ws-arch/.

[15] The OWL Services Coalition, "OWL-S: Semantic Markup for Web Services," 2003, URL: http://www.daml.org/services/owl-s/1.0/owl-s.html.

[16] W3C Recommendation, *SOAP Version 1.2 Primer*, W3C, 2003, URL: http://www.w3c.org/TR/ soap12-part0/.

[17] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, May 2001, URL: http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2.

[18] W3C Recommendation, *OWL Web Ontology Language Guide*, W3C, 2004, URL: http://www.w3c.org/TR/owl-guide/.

[19] W3C Recommendation, *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*, W3C, 2003, URL: http://www.w3.org/TR/wsdl20-primer/.

[20] W3C Recommendation, *RDF Primer*, W3C, 2004, URL: http://www.w3c.org/TR/rdf-primer/.

[21] L. Miller, A. Seaborne, and A. Reggiori, "Three Implementations of SquishQL, a Simple RDF Query Language," *Proc. 1st Int'l Semantic Web Conf. (ISWC 2002)*, LNCS 2342, Springer-Verlag, 2002, pp. 423-435.

[22] W3C Recommendation, *RDF Semantics*, W3C, 2004, URL: http://www.w3c.org/TR/rdf-mt/.

[23] W3C Recommendation, *RDF Vocabulary Description Language*, W3C, 2004, URL: http://www.w3c.org/TR/rdf-schema/.

[24] W3C Recommendation, *OWL Web Ontology Language Semantics and Abstract Syntax*, W3C, 2004, URL: http://www.w3c.org/TR/owl-absyn/.

[25] S. Narayanan and S. McIlraith, "Simulation, Verification and Automated Composition of Web Services," *Proc. Int'l World Wide Web Conference (WWW-11)*, Honolulu, Hawaii, May 2002, pp. 77-88.

[26] K. Erol, D. Nau, and J. Hendler, "UMCP: A Sound and Complete Planning Procedure for HTN Planning," *Proc. AIPS-94*, Chicago, 1994, pp. 249-254.

[27] E. Sirin, B. Parsia, and D. Wu et al., "HTN Planning for Web Service Composition Using SHOP2," *Web Semantics*, Elsevier, vol. 1, Issue 4, Oct. 2004, URL: http://www.mindswap.org/papers/SHOP-JWS.pdf.

[28] T. Andrews, F. Curbera, and H. Dholakia et al., "BPEL for Web Services," 2003, URL: http://www.ibm.com/developerworks/library/ws-bpel/.

[29] Evolution Robotics, *ERSP 3.0: Robotic Development Platform,* URL: http://www.evolution.com/products/ersp/.

[30] S. Jin, C. Park, and D. Choi et al., "Cluster-Based Trust Evaluation Scheme in an Ad Hoc Network," *ETRI J.*, vol. 27, no. 4, Aug. 2005, pp. 465-468.

Young-Guk Ha received the BS and MS degrees in computer science from Konkuk University in 1993 and 1995. He is currently working toward the PhD degree in computer science at Korea Advanced Institute of Science and Technology (KAIST). He joined Electronics and Telecommunications Research Institute (ETRI) in 1995 and has been a Senior Member of Engineering Staff in the Intelligent Robot Research Division since 2004. His research interests are in ubiquitous networking, sensor networks, and the semantic web technology.

Joo-Chan Sohn received the MS degree in management information systems from Hankook University of Foreign Studies in 1990. After joining ETRI, he has been involved with electronic commerce systems and intelligent e-business systems. Currently, he is a Senior Member of Engineering Staff in the Intelligent Robot Research Division and his research interests are in intelligent service infrastructure for robots and artificial emotion systems.

Young-Jo Cho received the BS degree in control engineering from Seoul National University in 1983, and the MS and PhD degrees in electrical engineering from KAIST in 1985 and 1989. He joined ETRI in 2004 as a Vice President heading the Intelligent Robot Research Division. Before joining ETRI, he was a Principal Researcher in Korea Institute of Science and Technology (KIST) and the Director of the R&D Center in iControls, Inc. His primary research interest is in network-based control architecture of intelligent mobile robots.

Hyunsoo Yoon received the BE degree in electronics engineering from Seoul National University in 1979, the MS degree in computer science from KAIST in 1981, and the PhD degree in computer and information science from Ohio State University in 1988. He was with AT&T Bell Labs from 1988 to 1989 as a Member of Technical Staff. Since 1989 he has been a Faculty Member of the Division of Computer Science at KAIST. His main research interests include wireless sensor networks, 4G networks, and network security.