INTECH
open science | open minds

OPEN ACCESS
ARTICLE

International Journal of Advanced Robotic Systems

A Service-Oriented Framework for the Development of Home Robots

Regular Paper

Tsung-Hsien Yang[1] and Wei-Po Lee[1,*]

1 Department of Information Management, National Sun Yat-sen University, Taiwan
* Corresponding author E-mail: wplee@mail.nsysu.edu.tw

Received 27 Jun 2012; Accepted 13 Nov 2012

DOI: 10.5772/55055

Abstract In recent years, researchers have been building home robots able to interact and work with people. Yet, because of the complicated and independent robot development environments, it is not always easy to share and reuse robot code created by different providers. In this work, we present an ontology-based framework that integrates service-oriented computing environments with the standard web interface to develop reusable robotic services. In addition to the service discovery, selection, and composition processes often performed by traditional web services, our work also includes an adaptive mechanism through which the user can iteratively modify composite robotic services to suit his or her needs. The proposed methodology has been implemented and evaluated, and the results show that our framework can be used to build robotic services successfully.

Keywords Service-Oriented Computing, AI Planning, Ontology, Service Composition, Robotic Service

1. Introduction

Developing home service robots to achieve user-specified tasks has become a significant issue with regards to how we will one day live our lives. Scientists and engineers are now building service robots that can interact and work with people in the home environment. However, due to the complexity and diversity of robot development environments, it is hard to integrate and share the robot application services constructed by different providers. This has therefore impeded the development of service robots [1-3].

To overcome the difficulties described above, researchers have been building robot design frameworks to manage the complexity and facilitate the reusability of robot code. From the end-users' perspective, they expect to obtain and use the robot application software conveniently without explicit programming. Robot designers prefer an easy-to-share environment in order to integrate application services constructed by different providers and concentrate on creating more advanced services. Considering the needs of both sides, service-oriented architecture (SOA, [4]) provides a promising choice for developing robotic services. One of the major advantages of SOA is that the shared resources are available on demand. This means that these resources can be regarded as independent services and accessed without knowledge of their underlying platform implementation [1,4]. Ideally, with an SOA-based robotic framework, end-users can control their robot in a similar way to that in which

they use web services. Moreover, robot designers can reuse available code to design more comprehensive services.

Yet, unlike traditional web services, applying SOA to robot applications involves complicated robot action control and the complexity of robot control increases along with the complexity of the task. In the development of robot code to solve application tasks, a common way to reduce complexity is to adopt a divide-and-conquer strategy. That is, to recursively break down a task into sub-tasks of the same type until the sub-tasks become simple enough to be solved directly and then to solve these sub-tasks in reverse order. The above concept of dividing and solving a robotic task is similar to the concept of configuring a composite service to accomplish a specific task in the web service domain, where a solution to the target task is composed of a set of simple services (each of which can solve a part of the original task) through a composition method [5]. Most composition problems can be solved by workflow-based or AI planning methods [6-7]. More recently, the semi-automatic service composition method has become popular, because it is in fact very difficult to compose the services in a fully automatic way. Hierarchical Task Network (HTN) planners are typical examples [5]. This kind of "hand-tailorable" planners is indicative of the current trend of combining automatic planners with human efforts to generate composite tasks.

In this work, we present a robotic service framework that tries to meet the needs of end-users and robot designers. Following the success of web services, our framework takes a service-oriented architecture in which a robot controller is created and regarded as a service and complicated robot tasks are effectuated successfully through the composition of available services. In addition, we also develop an adaptive mechanism with which to modify incorrect or unexpected robot action sequences during the service composition process. To implement the proposed methodology effectively, our work integrates the OWL-S and HTN planners to describe robotic services and perform service composition respectively. Unlike traditional web services, the robotic services here are stored in a publicly available repository, which will dispatch a duplication of the control code included in a service to the requester. To verify the effectiveness of the proposed methodology, we use it to achieve a user-specified control task in a home environment. The results show the promise of our work.

2. Related Work

Web service technologies have been widely applied to the design of web-based applications. With many successful experiences, researchers are now advocating the extension of these techniques to the development of robot systems. The most related works are the ones concerning robots as services and exploiting the web service architecture to create robots. For example, Yachir *et al.* employed service composition techniques to plan robotic services to help an elderly person [8]. Their work emphasized how one can decrease the number of parameters and services in the composition process to enhance efficiency. Kim *et al.* focused on controlling robots through the integration of web service and robot application technologies. They used a web service framework to provide functional communication for a robot and as a means to control the robot in performing pre-specified commands [9]. Similarly, Ha *et al.* proposed a service-oriented architecture for the integration of ubiquitous robot devices, including sensors and motors [10]. To find a suitable service, they developed an ontology-based method to interpret the task query specified by the user.

There are also other works which integrate robotic and web technologies. For example, Mokarizadeh *et al*. proposed a framework for using web service composition to plan actions among robots. A user on the client side can use the web as a means of transmission with which to control a robot for the completion of certain tasks (such as reading the temperature of a room) [11]. In a recent piece of research conducted by Blake *et al*., robots were equipped with web-oriented software interfaces that help them access universally standard web resources [12]. To enable knowledge acquisition and data reuse for robots, Waibel *et al.* developed an open-source platform RoboEarth to provide a networked database repository for robots to share information with others [13]. In addition, Osentoski and colleagues have built a framework which enables web interfaces for robots to share code among roboticists [14].

Ontology is a practical way to deliver semantics between users and systems and has previously been used in the successful transmission of domain knowledge. It has also been applied to robotic applications, mainly to conceptualize the robots (such as their internal structures and functions), their tasks, or the environments they are situated in. For example, Lim *et al*. developed an ontology-based knowledge framework to include low-level knowledge (the perceptions and actions of the robot) and high-level knowledge (the world model) [15]. To solve a spatial planning task, Belouaer *et al*. also used ontology to represent the robot environment [16]. In addition, Jeon *et al.* defined a domain ontology to parse and recognize the user's intentions for his or her robot behaviour planning [17].

Unlike the above studies, our work dedicates itself to constructing an explicit task ontology for daily home tasks and focuses on planning-based service composition to provide robotic services. In the construction of

ontology, the web services description language, the Ontology Web Language-Services (OWL-S, [18]), is used to describe service information and the relationships between different services to enrich the semantics for the services. Using a web-based standard platform and interface, the services created by different providers can thus be easily shared and integrated. Most importantly, we develop an adaptation mechanism with service re-planning to deal with undesired robot behaviours.

3. An Ontology-Based Robotic Service System

As proclaimed in [19], service-oriented computing (SOC) has become a contemporary programming paradigm. It regards services as self-described reusable building blocks that can be used to support the development of software applications. SOC applies SOA to organize software applications and infrastructure into a set of interacting services. Among others, web services are currently the most promising SOC-based technology. They use OWL-S to describe services in an unambiguous, computer-interpretable form. The process described above enables the automatic discovery, selection, composition and execution monitoring of services. In addition, the loosely coupled interoperations between services can be achieved by using the simple object access protocol (SOAP) or the representational state-transfer protocol (REST) to convey or deliver messages. Following the SOC design principles, in this work we present a service-oriented framework that is implemented with standard-based, platform-independent techniques. It can thus exploit the corresponding advantages of SOC to provide rapidly prototyping robotic services for end-users.

3.1 System overview

The conceptual framework and the operating flow of the proposed robotic service system are illustrated in Figure 1. As can be seen, our system is mainly comprised of three major components, a search agent (SA), a composition agent (CA) and an adaptation agent (AA). The SA is responsible for analyzing the control command (or query) specified by the user, so that the system can search and acquire robot services accordingly from the service repository. The task ontology shown in Figure 1 is built for command interpretation. It describes the semantics of a task in terms of the task structure and the task-solving process. The SA then searches the task ontology by directly mapping the vocabulary of the user command and matching it with the task terms recorded in the ontology. If the agent cannot find any corresponding task term for the command, it will then try to map the command into the cognitive synonyms (collected from the synsets of WordNet [20]) of the task instead.

The second agent, CA, is developed for service discovery, selection and composition. It retrieves the services which correspond to the task terms found by the SA from the service repository and employs a pre-defined selection strategy to choose the most suitable service from among the candidates. For some cases where the required robot services are not available (e.g., not provided by any developer), the CA will decompose the original task into several subtasks in a recursive manner according to the specification of the task ontology and will find relevant services for each subtask. Then it will further integrate these services by means of a composition procedure to achieve the target task. The third agent, AA, has the role of system evaluation and adaptation. It presents the action-planning steps to the user and makes a simulated/real robot perform the services as delivered by the agent CA. The user can evaluate the robot behaviour and indicate if any behaviour sequences are incorrect (or unexpected) through a pre-designed interface. Agent AA will send the correction list back to the CA and ask it to start a modification procedure to find new services to replace the ones identified as presenting incorrect behaviours. The details are described in the sections below.

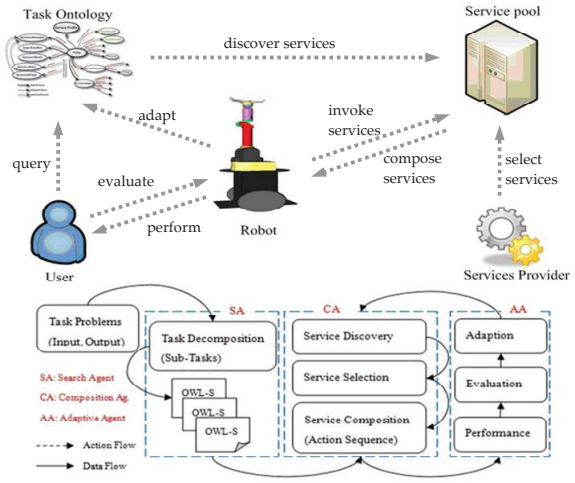

Figure 1. System overview.

3.2 Knowledge ontology for problem solving

As can be observed in the above section, ontology plays an important role in the mutual-understanding between users and the service system. It has been used for knowledge representation and inference in the artificial intelligence research community for many years. Though ontology can clarify the structure of knowledge and enable knowledge sharing, it is not task-dependent. In fact, it has been indicated that in building a problem-solver, a designer needs both domain factual knowledge and problem-solving knowledge [21]. For an action planning problem (such as solving a robot task), ontology can be defined so as to provide the sequence of problem-solving steps through organizing domain knowledge. The problem-solver can then exploit domain knowledge and the related techniques defined in the ontology to achieve the effective implementation of the application task.

Therefore, in this work we develop two ontologies, task ontology and position ontology, to specify the structure of the problem-solving process and to describe the environmental knowledge for the robot, respectively. To accomplish the target task in a home environment, a service robot should understand both ontologies as described as follows:

- Position ontology: This ontology defines the locations of different objects in a home environment and the containment relationships between the objects. Figure 2 shows a part of the position ontology. It illustrates that the bowls, plates, and cups are put on a cupboard, and a TV is placed in the living room.
- Task ontology: This ontology shows how to resolve a user's request using a sequence of steps. It presents the complexity of the tasks hierarchically. Figure 3 shows a part of the task ontology as defined in the proposed system. The task ontology describes the possible actions for the robot (i.e., what the robot can perform) in the home environment. After receiving the control command from the user, the system will parse the command to extract the verb as the action description and then match the description with the task terms recorded in the task ontology. For example, the command "give me a cup" is firstly parsed to obtain the verb "give". Then the word "give" is matched with the terms included in the task ontology.

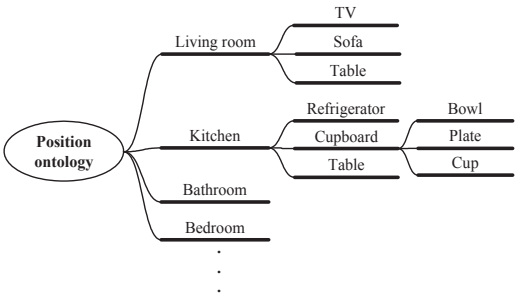

Figure 2. Part of the position ontology.

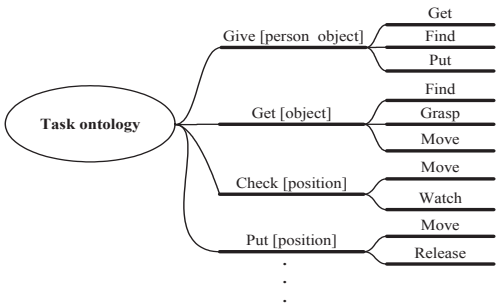

Figure 3. Part of the task ontology.

In order to enhance the problem-solving capability of the task ontology, an efficient and powerful ontology description language with semantics is needed. OWL-S provides an ideal choice to implement the ontology needed in our robotic service system. It has three major parts with essential types of knowledge: service profile, service model and service grounding. In general, the profile provides the information to express "what the service does", including a description of what needs to be accomplished and the limitations and requirements of the service. The service model describes "how to use the service", including how to perform an analysis (to examine whether the service meets the user's needs), a composition (to compose multiple services to achieve a specific goal), activities (to coordinate the activities of different participants) and monitoring (to inspect the execution of the service). The service grounding presents "how to access a service". Its role is to specify a message format, a communication protocol and other implementation details. Our system uses OWL-S for ontology construction and it mainly uses service profile for selection and service model for composition. In particular, the service model shows how to use the service step by step, which is similar to the problem solving process for annotating robotic services.

3.3 Service discovery, selection and composition

The ontologies defined above can enable different types of service functions, including automatic service discovery, selection and composition. Service discovery is the process of searching available services to find ones to satisfy the user's requests, based on the description of the functional and non-functional semantics provided by the user. As mentioned, using OWL-S to describe services can standardize the service description in semantics. In this way, an autonomous agent can be developed to find appropriate services, according to the semantic descriptions associated with the services. Here, we use this method to define and annotate robot services and construct a search agent responsible for service discovery. In our system, the OWL-S service profile defines the IOPE elements (i.e., Input, Output, Precondition and Effect), which are then later used to match the user's service requests with services. The procedure is that if the control command can find a match for the task terms defined in the ontology, the system will collect the relevant information (i.e., IOPE) corresponding to this task term to examine whether the information matches that of the services available in the service repository. A successful match means that the system can find services to fulfil the user's request. For instance, in the task ontology shown in Figure 3, the task "Get" has the input "target object name", the precondition "target object exists" and the effect "target object is on hand". Once the task "Get" matches the user command, the task descriptions are used to find services.

The second type of functions, service selection, is the process of choosing the most suitable services for the target task from the candidates provided by the service

discovery process. To select the best service, many researchers have proposed different QoS-based selection methods. The attributes often sought after in the QoS methods include accessibility, availability, cost, response time, error rate, throughput, reliability, reputation, robustness and security [22-23]. Among these attributes, service reputation directly reflects the results of the evaluation by the service requesters or the neutral party. It is an objective and easy-to-measure attribute. Therefore, in our current implementation, we choose to use this attribute for service selection and design a rating strategy to measure service reputation.

Our strategy is based on the concept of collaborative filtering approach ([24-25]) often used in the application of product recommendations. In its original form, the collaborative approach is to recommend items to a user based on the evaluation results gained from other users with similar tastes. It first measures the similarity between users by a certain correlation criterion and then employs a *k*-nearest neighbour method to find the most similar users to perform a recommendation. The prediction of an unknown item for a user is thus based on the combination of the ratings of his or her nearest neighbours. At present, we take all users' opinions into consideration and do not conduct similarity measurement. The strategy can be easily extended to collect similar users if necessary.

In our system, users are allowed to evaluate and rate the services they have consumed. The rating value, which is an integer ranging from 1 (lowest) to 5 (highest), indicates the user's level of satisfaction. All ratings given by users are combined to rank the candidate services obtained from the service discovery process. The system will then select services for the user, according to the ranking order.

If the system finds no suitable services from the above procedures, it will start a further service composition process to find composite solutions for the target task. In the proposed framework, we adopt the AI planning techniques to compose services already in the repository. Among others, HTN planning is a well-designed methodology suitable for service composition for several reasons. For example, it encourages modularity that fits in well the SOA, it can easily scale up to large numbers of services and it has means by which to minimize various sorts of failures or costs, because the planner considers the entire execution path. Further analysis and the corresponding advantages of using HTN for service composition are referred to in reference [5].

In the HTN planning domain, the tasks can be categorized into two types: the primitive or the compound. A primitive task can be performed directly by the predefined planning *operators*, while a compound task needs to be decomposed by a planning *method*. It performs task decomposition to break down the original task from the complex (high level) down to the simple (low level) hierarchically. With the results of the decomposition the planner solves the subtasks in reverse order and can produce a sequence of actions for the original task.

The concept of task decomposition in HTN is very similar to the composite process decomposition in the OWL-S service model. It reduces the complexity of reasoning by eliminating uncertainty during the planning process. One well-implemented HTN planner is the Simple Hierarchical Ordered Planner 2 (SHOP2, [26]). The authors of reference [5] have proved the semantic correspondences between the SHOP2 planning and the situation calculus of the OWL-S process model. They have also shown that SHOP2 can be used to compose web services effectively. This means that the HTN planner is a suitable tool to work with the hierarchically structured OWL-S process model. Following their successful studies, we choose to employ SHOP2 to conduct our robotic service composition.

OWL-S atomic	hasInput	has Precondition	hasResult (negative)	hasResult (positive)
SHOP2 operator	*Input parameters*	*Pre*	*Del*	*Add*
OWL-S composite	hasInput	has Precondition	Control construct	List of atomic & composite
SHOP2 method	*Input parameters*	*Pre*	π	*Task*

Translate-atomic-process-effect(*Q*)
Input: an OWL-S definition *Q* of an atomic process *A* with only effects.
Output: a SHOP2 operator *O*.
$\vec{v}$ = the list of input parameters defined for *A* in *Q*.
Pre = conjunct of all preconditions of *A* as defined in *Q*.
Add = collection of all positive effects of *A* as defined in *Q*.
Del = collection of all negative effects of *A* as defined in *Q*.
Return $O = (A(\vec{v})\ Pre\ Del\ Add)$.

Translate-If-Then-Else-Process(*Q*)
Input: an OWL-S definition *Q* of a composite process *C* with *If-Then-Else* control construct.
Output: a SHOP2 method *M*.
$\vec{v}$ = the list of input parameters defined for *C* in *Q*.
π_{if} = conditions for *If* as defined in *Q*.
Pre_1 = conjunct of all preconditions of *C* in *Q* and π_{if}.
Pre_2 = conjunct of all preconditions of *C* as defined in *Q*.
$Task_1$= process for *Then* as defined in *Q*.
$Task_2$= process for *Else* as defined in *Q*.
Return $M = (C(\vec{v})\ Pre_1\ Task_1\ Pre_2\ Task_2)$.

Table 1. Two examples of OWL-S and SHOP2 translations.

As OWL-S and SHOP2 have their own internal representations for information processing, in order to use SHOP2 for service composition, some translations between OWL-S and SHOP2 have been defined in [5]. These translations are from the OWL-S process models to the SHOP2 domains and from the OWL-S composition

tasks to the SHOP2 planning problems. Table 1 (the upper part) shows the translations for the planning elements, *operator* and *method* mentioned above. To explain such translations in detail, this table (the lower part) also lists two major translations: one is the translation of the atomic OWL-S definition "atomic process" into a planning *operator*, and the other is the translation of an OWL-S process (with the *If-Then-Else* control construct) into a planning *method*. With such transitions, a service originally created by the OWL-S descriptions can have its SHOP2 format and be used by the planner for service composition.

3.4 System adaptation

After having used the service discovery, selection and composition processes described in section 3.3, the user can expect a set of services to complete his or her task. However, due to unforeseen situations, in some cases the robot cannot achieve the task with the services organized by the planner. To remedy such a failure, the system needs to consider the newly acquired world states and adapt to these states by performing a re-planning procedure iteratively. As per our design, the system will highlight the planning steps which correspond to the incorrect behaviour sequences. The user can inspect the highlighted part to see what caused the failure, modify the world states accordingly (i.e., the preconditions that need to be satisfied but are not yet listed in the available services) and then activate the planner again in order to find other services to replace the current ones. Through the above interactive procedure for feedback collection, the system can gradually derive services that most meets the user's needs.

To effectively implement system adaptation (i.e., iterative re-planning), we use the software OpenRAVE (Open Robotics and Animation Virtual Environment, [27]) to construct simulated robots and the environments the robots are situated in. OpenRAVE is an open-source cross-platform software architecture. It provides high-level scripting, motion planning, perception and tests control algorithms. As this software has focused on designing high-level robotic tasks rather than low-level control, users can thus concentrate on the development of autonomous motion planning without being distracted by operating details (for example, collision detection). In this way, robot developers can easily share and compare different controllers or algorithms with others. As high-level scripting language is helpful for object-oriented design and useful in invoking service functions, we choose therefore to use the high-level language Python to implement the proposed method.

As shown in Figure 4, when the system executes a composite service in OpenRAVE, the planned action steps are also presented in the robotic service interface. As it is, the system is designed to present the solution (i.e., the sequence of steps) which is most highly recommended by the planner. It can also be modified to list multiple solutions for the users' reference. With the results thus shown in the interface, the user can compare the planned steps with the robot actions to determine whether the robot has achieved the task successfully. If the user finds that the actions in the simulation do not match his or her request, he or she can mark the wrongly performed steps and ask the system to conduct the service composition again. The user can also create new services to fix an incomplete plan through a demonstration-based mechanism we developed previously [28], in which the user can generate control code without explicit programming.

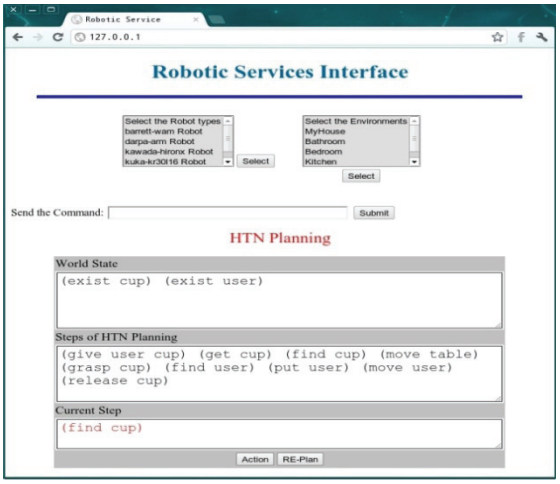

Figure 4. The interface for showing the current world states and planning steps.

4. Implementations and Experiments

As indicated in the first section, this work aims to provide an integrative approach to meet the needs of end-users and robot designers. Because web services are based on common industry standards and existing technology, applications that rely on web services can be deployed easily regardless of the language, platform, or internal protocols they use. The proposed framework has adopted the same design methodology, so that new robotics services can easily be shared and reused. In this section, we will concentrate on presenting how our framework works with the use of a walk-through example.

4.1 Application task

To verify the proposed methodology, we use it to carry out a simulation of an application task, in which a home robot deals with daily chores for its owner. The robot has some pre-programmed basic service components (i.e., controllers), and the user needs to develop new controllers for more complicated tasks. The application task is that the user asks the robot to give him a cup. To understand the user's request, the robot needs to connect

to the ontology server in order to parse the command "give me a cup". Because the robot has not experienced this task before, the system then tries to find the services directly or to decompose the task and find services for the subtasks accordingly if necessary. For this application task, the services related to the subtasks derived by the planner include "locate a cup", "navigate to the cup position", "grasp the cup", "locate the user", "navigate to the user", and "give the cup to the user". During the service execution period, the system will continuously use the changed world states to examine whether the robot has achieved the sub-goal at each stage. As described above, should the robot not be able to accomplish the overall task successfully, the system will present the planning steps with their corresponding world states (related to the subtasks unable to be achieved). Then the user is asked to provide more information in order to make a new plan. After the task is completed, the user is allowed to rate the services.

Two test scenarios were arranged for this task. In the first scenario, the cup is placed on the table and it is visible to the robot. This is mainly to verify whether the system can find a suitable service to drive the robot to pick up the cup. In the second scenario, the cup is moved into an opaque cupboard so that the robot needs to find the cup's location first and then pick it up. This case is to examine whether the system can adapt to a new environment and make a new plan to fulfil the target task.

4.2 System workflow

In our work, the user needs to select the robot type and the task environment from the pre-defined interface (as shown in Figure 4), before sending a control command to request robotic services. The robot type is used to derive some hard constraints (e.g., hardware restrictions) that need to be satisfied in the service discovery process. The task environment provides the initial world states (that are the preconditions of the desired service) and indicates the initial positions of the objects in the environment. In addition to the pre-defined choices, users are allowed to take the XML descriptions to create their own simulated robots and environments.

The system can be used to solve the application task described in the above section. After receiving the user command "give me a cup", the system uses a common natural-language parser ([29]) to check the syntax of the command and generate a syntax tree (as shown in Figure 5). It then sends the verb part to the task ontology and the noun part to the position ontology to search for suitable services. In this work, an exact-match scheme is used to perform direct vocabulary-mapping between the user command and the task terms recorded in the ontology. To stretch the scope of task terms in semantics, we have collected the synonyms for the task terms from the WordNet to form a word set in advance. If the system cannot find an exact task term to match the user's command, it will turn to the word set to find an identical match for the command.

As mentioned in section 3.2, the task ontology records the parameters for implementing a specific task, so the system can use these parameters to discover relevant services. In the part of the process illustrated in Figure 3, we can see that to achieve the "Give" task, the system needs two input parameters, namely "Person" and "Object", and it takes the segments "me" and "cup" extracted from the command to correspond to the two parameters, respectively. After the service discovery process, the system obtains some service candidates and it uses the selection strategy to evaluate and rank the candidates. Finally, a composite service "Give" with the highest reputation value is selected.

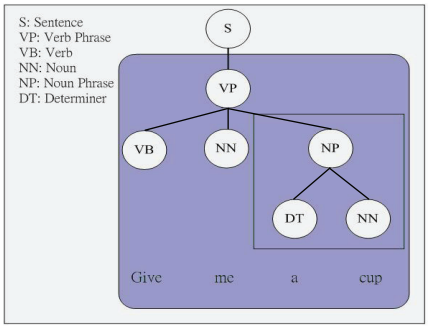

Figure 5. Parse tree of the user command.

The OWL-S example shown in Figure 6 illustrates a typical service model used to solve robot tasks. The description of the model is similar to the ones for web service applications. It mainly includes information about inputs, outputs, preconditions and effects of a service, so that the service requester can perform an analysis, composition, activities and monitoring to achieve a specific goal. The OWL-S process ontology provides different markups (such as CompoiteProcess, composedOf, Sequence, etc. shown in the example) to model services as processes. Further details on service modelling are referred to in reference [18]. Robotic service providers can follow the relevant instructions to create new services.

This example describes the model as defined for the robot behaviour, "Give", which is composed of one atomic process, "Find", and two composite processes, "Get" and "Put". As can be seen, the task "Give an object to a person" can be broken down into three subtasks: "Get the object", "Find the person", and "Put the object to the person (meaning location)". In addition, the process for "Give" has two inputs "Person" and "Object" (to tell the robot which object to take, and to whom) and two preconditions "ExistPerson" and "ExistObject" (to indicate if any person or object exists in the world state).

```
<process:CompositeProcess rdf:ID="Give">
  <process:hasInput rdf:resource="#Person"/>
  <process:hasInput rdf:resource="#Object"/>
  <process:hasPreconditoion rdf:resource="#ExistPerson"/>
  <process:hasPreconditoion rdf:resource="#ExistObject"/>
  <process:hasEffect rdf:resource="#PersonHasObject"/>
  <process:composedOF>
    <process:Sequence>
      <process:components rdf:parseType="Collection">
        <process:CompositeProcess rdf:about="#Get"/>
        <process:AtomicProcess rdf:about="#Find"/>
        <process:CompositeProcess rdf:about="#Put"/>
      </process:components>
    </process:Sequence>
  </process:composedOf>
</process:CompositeProcess>
```

Figure 6. The OWL-S example.

With results of decomposition, the system can then translate the "Give" service described by OWL-S into a HTN *method* and invoke the sub-services included in the "Give" process to complete the overall task. Figure 7 (a) shows the HTN *method* which corresponds to the "Give" service, which includes the subtasks "Get", "Find" and "Put". The action steps of "Give me a cup" can be carried out by the relevant atomic services (i.e., HTN *operators*), in the order of (find cup), (move table), (grasp cup), (find user), (move user) and (release cup).

However, in the second test scenario (as described in section 4.1) in which the cup is hidden in an opaque cupboard, the robot cannot grasp the cup directly, although it can obtain the cup's position from the loaded environment file and then move to a position around the cup. This means that the current service can not satisfy the user's requirement. To cope with such a failure, the system highlights the relevant steps in the HTN plan, so that the user can inspect this part so as to find out the reasons for the failure, modify the world state and make a new plan.

In the experiment, the system highlighted that the robot could not complete the subtask "grasp". By examining the robot actions corresponding to the highlighted part (the problematic part), the user found that the cup did not appear in a visible place so the robot could not grasp it. To solve this problem, the user added a new world state "(isContainer cupboard)" as the precondition and a new input "cupboard" to the planner to find suitable services. After that, the system found a "search" service (to open containers to check if any of them contains the target object) that matched the new requirement. The new plan also indicated that this "search" service must be performed before the "grasp" service (see Figure 7(b) and (c)).

Taking the new plan, the robot could first check the position ontology to obtain the possible cup position and it then checked the world states to find all the containers which could be used to store the cup. After checking the position ontology, the robot realized that the cup was placed in the cupboard and it confirmed that the world state "(isContainer cupboard)" was true. In line with the other world state "(isClosed cupboard)", the robot tried to open the cupboard, and then grasped the cup successfully.

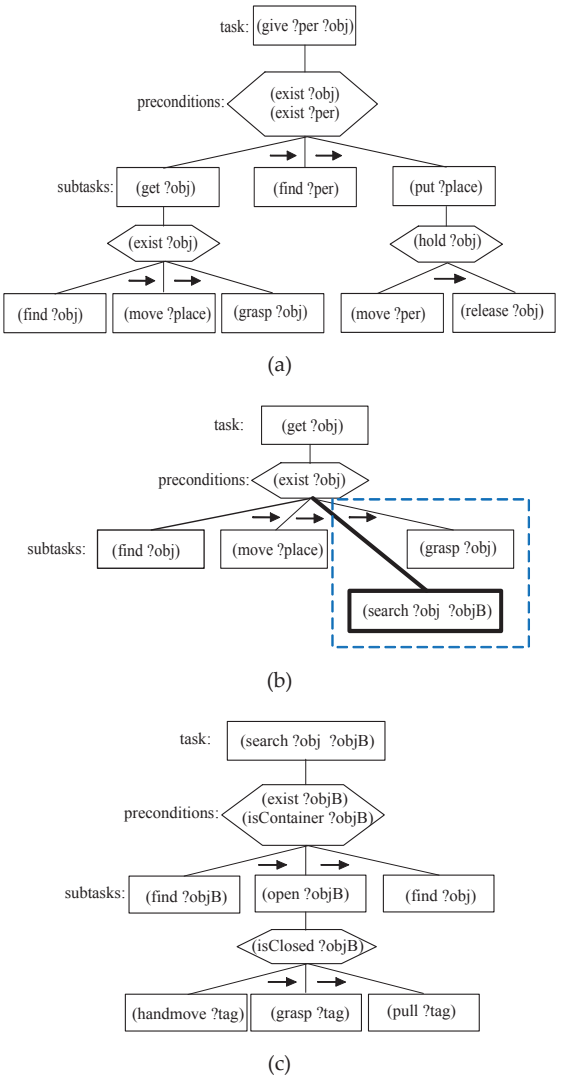

Figure 7. The planning and re-planning.

4.3 Simulation results

Figure 8 illustrates how the robot achieved the task in the two test scenarios which were conducted in a simulation. The first four steps in the figure (i.e., steps (1)-(4)) show that the robot used the position ontology to infer where the cup was located. In the first simulation, once the robot knew that the cup was in the kitchen, it went there and recognized that the cup was put on the table. Then the robot moved to the region around the cup so that it was able to grasp the cup. Steps (5)-(7) in Figure 8 describe the following: the robot moved to a position close to the cup, and performed the "Get" service. Next, steps (8)-(12) show that after the robot had picked up the cup, it invoked the other two services "Find" and "Put" to find the user and give him or her the cup.

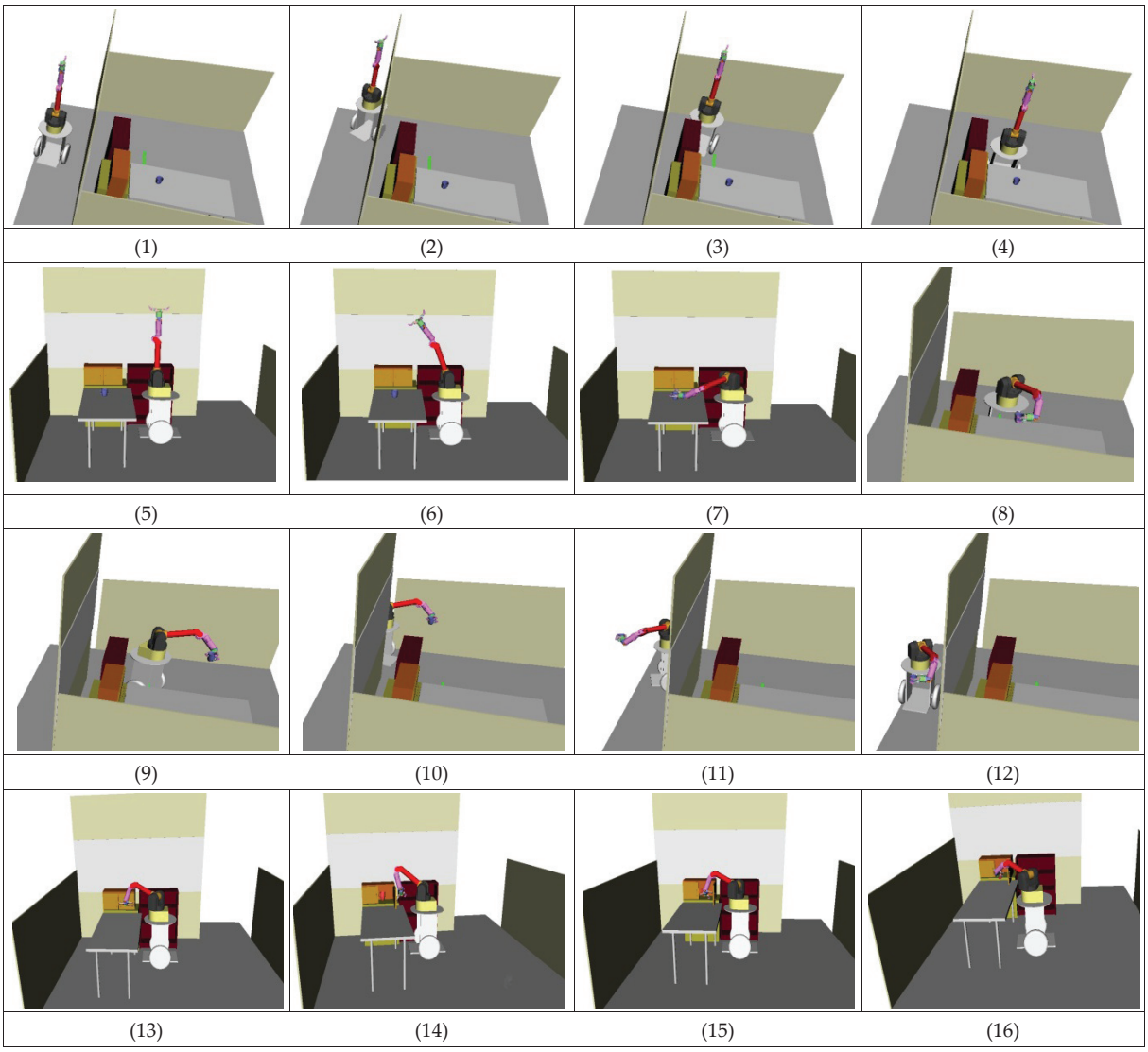

Figure 8. The simulation results for the application task.

As mentioned before, if the robot cannot complete the task with the services obtained from the composition process, the system needs to start a re-planning procedure to find other services for the robot to perform. In the second task scenario here, the initial plan did not work, because the service "Find", with the environment being as it is, cannot result in any state "found object" which was the precondition of the service followed. To deal with this failure, the user asked the system to change the world state from "cup is visible" to "cup is invisible", and then to use the HTN planner again to make a new plan. This time, a new service "Search" was found and used to seek the object's "containers". For any object with a property "container", the robot activated the service "Open" to open it, and checked if the cup was inside. Steps (13)-(16) of Figure 8 represent to the process described above, in which the robot opened the cupboard and then grasped the cup to achieve the task.

5. Conclusions

In this work, we have emphasized the importance of developing an easy-to-share networking platform for the reuse of robot code distributed by different providers. Considering the needs of both sides, robot designers and end-users, we presented an ontology-based framework that integrates a service-oriented computing environment with the standard web interface to facilitate the sharing of robotic services. In addition to the service discovery and selection processes, our framework includes a planning-based composition mechanism to generate composite robotic services to solve complicated tasks. It also performs 'system adaptation' to continuously update world states and make new plans. To verify the proposed methodology, experiments were conducted, and the results show that a composite service can be derived for the application task successfully.

As can be observed, the proposed robotic service framework has some advantages. First, with the help of task ontology, users can share their problem-solving concepts and reuse their individually-designed services. Second, our work follows the SOC principles and thus features high-level interoperability and simplified

services integration. Third, the specially-designed system adaptation can increase system stability and user satisfaction in the changed environment.

Based on the presented framework, we are now extending our work in several directions. To enrich the task ontology, we plan to include other domestic tasks defined in relevant ontology studies to construct a more complete ontology. Meanwhile, it is worthwhile exploring new ways to tackle the scalability problem when more and more services are required to solve tasks with high complexity.

6. Acknowledgments

This work has been supported by the National Science Council of Taiwan (Grant no. NSC 101-2221-E-110-094-).

7. References

[1] Amoretti M, Reggiani M (2010) Architectural Paradigms for Robotics Applications. *Advanced Engineering Informatics*, 24(1): 4-13.

[2] Bruyninckx H (2008) Robot Software: the Future Should Be Open. *IEEE Robotics and Automation Magazine*, 15(1): 9-11.

[3] Remy S, Blake MB (2011) Distributed Service-Oriented Robotics. *IEEE Internet Computing*, 15(2): 70-74.

[4] Erl T (2005) *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall.

[5] Sirin E, Parsia B, Wu D, Hendler J, Nau D (2004) HTN Planning for Web Service Composition Using SHOP2. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(4): 377-396.

[6] Rao J, Su X (2005) A Survey of Automated Web Service Composition Methods. *Semantic Web Services and Web Process Composition*, pp.43-54.

[7] Lecue F, Mehandjiev N (2011) Seeking Quality of Web Service Composition in a Semantic Dimension. *IEEE Trans. on Knowledge and Data Engineering*, 23(5): 942-959.

[8] Yachir A, Tari K, Chibani A, Amirat Y (2008) Towards an Automatic Approach for Ubiquitous Robotic Services Composition. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3717-3724.

[9] Kim BK, Miyazaki M, Ohba K, Hirai S, Tanie K (2005) Web Services Based Robot Control Platform for Ubiquitous Functions. *Proceedings of IEEE International Conference on Robotic and Automation*, pp. 691-696.

[10] Ha YG, Sohn JC, Cho YJ, Yoon H (2007) A Robotic Service Framework Supporting Automated Integration of Ubiquitous Sensors and Devices. *Information Sciences*, 177(3): 657-679.

[11] Mokarizadeh S, *et al.* (2009) Applying Semantic Web Service Composition for Action Planning in Multi-robot Systems. *Proceedings of the Fourth International Conference on Internet and Web Applications and Services*, pp. 370-376.

[12] Blake MB, Remy SL, Wei Y, Howard AM (2011) Robots on the Web. *IEEE Robotics and Automation Magazine*, 18(2): 33-43.

[13] Waibel M, *et al.* (2011) RoboEarth: A World Wide Web for Robots. *IEEE Robotics and Automation Magazine*, 18(2): 69-82.

[14] Osentoski S, *et al.* (2011) Robots as Web Services: Reproducible Experimentation and Application Development Using rosjs. *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 6078-6083.

[15] Lim GH, Suh IH, Suh H (2011) Ontology-Based Unified Robot Knowledge for Service Robots in Indoor Environments. *IEEE Trans. on Systems, Man, and Cybernetics, Part A* , 41(3), 492-509.

[16] Belouaer L, Bouzid M, Mouaddib A (2010) Ontology Based Spatial Plannng for Human-Robot Interaction. *Proceedings of the 17th International Symposium on Temporal Representation and Reasoning*, pp. 103-110.

[17] Jeon H, Kim T, Choi J (2008) Ontology-Based User Intention Recognition for Proactive Planning of Intelligent Robot Behavior. *Proceedings of International Conference on Multimedia and Ubiquitous Engineering*, pp. 244-248.

[18] Martin D, *et al.* (2004) OWL-S: Semantic Markup for Web Services. *W3C Member Submission*, 22: 2007-04.

[19] Papazoglou MP, Traverso P, Dustdar S, Leymann F (2007). Service-Oriented Computing: States of the Art and Research Challenges. *IEEE Computer*, 40(11): 38-45.

[20] Fellbaum C (2010) WordNet. In: Poli R, Healy M, Kameas A, editors. *Theory and Applications of Ontology: Computer Applications*, pp. 231-243.

[21] Chandrasekaran B, *et al.* (1998) The Ontology of Tasks and Methods. *Proceedings of the Eleventh Banff Knowledge Acquisition for Knowledge-Based System Workshop*, pp. 18-23.

[22] Yu T, Zhang Y, Lin KJ (2007) Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints. *ACM Trans. on the Web*, 1(1), article number 6.

[23] Michlmayr A, Rosenberg F, Leitner P, Dustdar S (2010) End-to-End Support for QoS-Aware Service Selection, Binding, and Mediation in VRESCo. *IEEE Trans. on Service Computing*, 3(3): 193-205.

[24] Ricci F, Rokach L, Shapira B (2011) Introduction to Recommender Systems Handbook. In: Ricci F, Rokach L, Shapira B, Kantor P (eds) *Recommender Systems Handbook*. Springer, Berlin, pp 1-35.

[25] Schafer JB, Frankowski D, Herlocker J, Sen S (2007) Collaborative Filtering Recommender Systems. In: Brusilovsky P, Kobsa A, Nejdl W (eds) *The Adaptive Web: Methods and Strategies of Web Personalization*. Lecture Notes in Computer Science, Vol. 4321. Springer-Verlag, pp. 291-324.

[26] Nau D, *et al.* (2003) SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research*, 20: 379-404.

[27] Diankov R, Kuffner J (2008) OpenRAVE: a Planning Architecture for Autonomous Robotics, Technical Report CMU-RI-TR-08-34, Robotics Institute, Carnegie Mellon University.

[28] Lee WP, Yang TH (2011) Combining GRN Modeling and Demonstration-Based Programming for Robot Control. *Neural Computing and Applications*, 20(6): 909-921.

[29] Klein D, Manning CD (2003) Accurate Unlexicalized Parsing. *Proceedings of the 41th Annual Meetings for Computational Linguistics*, pp. 423-430.