A Service Intermediary Agent Framework for Web Service Integration

Jian Cao
Dept. of Computer Science and Engineering, Shanghai, China
Cao-jian@cs.sjtu.edu.cn

Jie Wang
Dept. of Civil and Environment Engineering, Stanford, USA
Jiewang@stanford.edu

Liang Hu
Dept. of Computer Science and Engineering, Shanghai, China
milkrain@qq.com

Rujie Lai
Dept. of Computer Science and Engineering, Shanghai, China
rujielaisusan@gmail.com

***Abstract*—Because web services and agent technology can help each other, to integrate them together is becoming a trend. Most of current research works focus on developing an integration framework to provide gateways that can connect the agent and web service worlds. In the paper, the service intermediary agent is proposed as a value-added professional service provider, which can manage a group of inherently related Web services. It organizes Web services as a set of plans and reacts to the requests by selecting and executing the appropriate plans. In order to process the incoming requests intelligently, the semantic goal structure is employed to facilitate plan organizing and plan selecting. The structure of the service intermediary agent and its working process are given. The semantic goal structure together with the deliberation process is given. The case study and experiments are also provided.**

Keywords-Web Service; Service Intemediary Agent; Plan

I. INTRODUCTION

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network [1]. It is based on a set of well- recognized standards like SOAP [2] and WSDL [3]. The Web service is expected to bring forth the next paradigm for the construction of information system, which relies on dynamically service discovering and composing. Unfortunately, current web service technology cannot realize this vision yet. Some deficiencies are: (1) It is very difficult to search an appropriate web service in terms of the requirements and understand its specification automatically in the open environment; (2) The web service cannot manage its behavior in an autonomic way [4]. For example, it cannot self-optimize its behaviors; (3) The web service can not differentiate requirements to offer customized service [5]; (4) Web services cannot collaborate with each other directly.

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators [6]. The concept of agent is now broadly used not only as a model for computer programming units but also in a more abstract and general way, as a new metaphor for the analysis, specification, and implementation of a complex software system.

To integrate web services and agent has becoming an important research direction for both agent technology and web service technology [7]. A web service should be able to invoke an agent service and vice versa. Some papers proposed a web service should be developed as an agent, for example, a tool developed by MTA SZTAKI can transform a web service into an agent [8]. However, the agent generated from a web service only has limited functions [9]. In other work, an agent is deployed as a web service when service endpoint is added [10][11]. But this approach only adds another communication port to the agent and exposing agent capabilities as a fixed set of services also restricts the agent reactiveness in some degree.

In this paper, a framework called Service Intermediary Agent (SIA) is proposed as a way to integrate web services for providing value-added professional services. It can select or compose appropriate web services to respond to the outside requests not only from the functional view but also from the qualitative view. The SIA's inherent knowledge base can be expanded and optimized so that its capability can be improved continuously. In addition, SIAs can cooperate with each other to satisfy the complex requirements. From the architectural view, the SIA is highly scalable since it can manage a large set of web services. We believe SIAs can form a common infrastructure to support web service collaboration in the future service world. This paper discussed how a SIA could organize the web services to react to requests accordingly.

This paper is organized as following. Section 2 briefly introduces the concept and the structure of a SIA. The execution mechanism of the SIA is explained in Section 3. The plan model for web service integration is introduced in Section 4. Section 5 discusses the way of plan selection for a request. Section 6 presents a case study and discusses some implementation issues. Section 7 introduces the related work and finally, Section 8 concludes the whole paper.

II. THE SERVICE INTERMEDIARY AGENT MODEL

With the prevailing of web services, it is anticipated that an IT system can be constructed by discovering and composing web services on demand. But web service discovery and composition are difficult tasks because special knowledge is needed during this process.

978-0-7695-4897-5/12 $26.00 © 2012 IEEE
DOI 10.1109/APSCC.2012.9

A SIA can offer value-added professional services since it has been equipped with the knowledge to organize, select and invoke it's managed web services. For example, an air ticket booking SIA knows how to book a ticket from various air companies through web service invocations and it also knows how to select one flight according to a passenger's cost requirements and other preferences.

Building an intelligent software agent is a difficult and time-consuming task that requires an understanding of advanced technologies such as knowledge representation, reasoning method, network communication methods and protocols, etc. Some companies or organizations delivered agent development tools such as JADE [10] and Agent Builder [12]. Although these tools can also help develop agents for web service integration, great manual programming efforts are still required because they do not provide any direct supports for web service integration.

Fig.1 shows the inherent structure of a SIA. It is based on BDI (Belief, Desire and Intention) concept [13], which is a software model developed for programming intelligent agents. In the SIA, beliefs represent the current states of an agent's internal and external worlds and are updated as new information about the worlds is received. A desire is a goal that the agent tries to perform some actions to achieve. And an intention is an agreement to associate the desire with behaviors, which is further be defined as a set of actions or plans.

According to the BDI concept, the structure of the SIA is divided into three parts: a Belief Model, a Control Model and an Execution System.

The belief model can be further divided into a World Model (WM), a Neighbor Model (NM), a Constraint Model (CM) and a Service Model (SM). The WM records the agent's inherent and environment states. The NM defines the other ones this agent will cooperate with. The CM consists of a set of constraints, which should be maintained by this SIA. The SM maintains a set of internal functional components and outside web services which are all regarded as different types of services.

The world model can be defined as $WM=<WS, WI>$, where WS stands for the data structure of the world model and WI is the instance of this data structure. WI can be further divided into GWI and $LWIS$, where GWI stands for the global world instance and $LWIS$ includes a set of session world instances. Each local session world instance is associated with a session that is created for a specific task. Therefore, the $LWIS$ is defined as $\{<LWI_i, Session_i>| i=1, 2, \ldots\}$. A new session can be created when the agent receives a request that consists of a serious of interactions and be destroyed when this interaction is completed. In this way, a SIA can process multiple interactive requests concurrently.

SIA has a goal model defined in its structure. The plan library includes a set of plans, which define the behaviors that can achieve the goal. Web services are integrated into the SIA as plans.

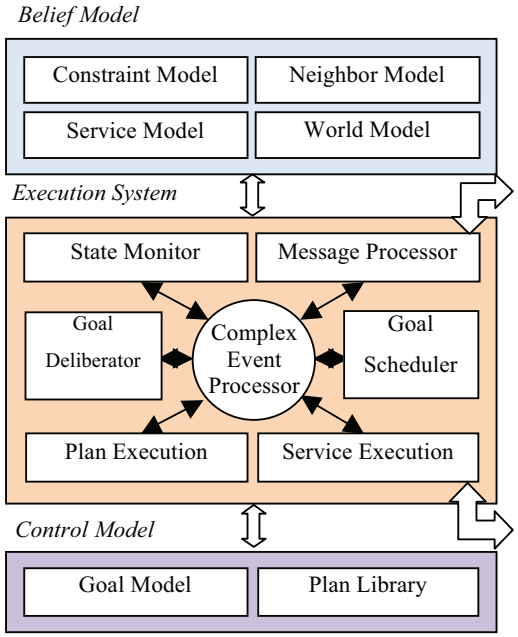

Figure 1. The Service Intermediary Agent Structure

The execution of the SIA relies on an execution system that is composed of several sub-components. The detail is introduced in the next Section.

III. The Working Mechanism of the Service Intermediary Agent

As an agent, the SIA should monitor the inside and outside changes, make decisions and take actions accordingly. The SIA takes advantage of a goal-oriented model to support its execution. And in order to make the SIA scalable, ontology is adopted for goal modeling.

Ontology is an explicit specification of a conceptualization and its importance has been well recognized [14]. The ontology applied in the SIA is structured as a set of individual generalization hierarchy terminology trees, with the more abstract concepts of the ontology forming the root terms of which other terms are specified. Each term of the hierarchy may be associated with a number of named attributes. Attributes are specified with an attribute name and type. Examples of built-in primitive types include Boolean, string, byte, integer, and real number. The complex types can be terms defined in other term trees. Attributes of a term are inherited by all of its children, which may have additional attributes.

If a term a is inherited from a term b, a specializes b and it is denoted as $a \in SP_t(b)$. Accordingly, b generalizes a and it is denoted as $b \in GE_t(a)$. The relationships of specialization (or generalization) are transitive. For example, if $c \in SP_t(a)$ and $a \in SP_t(b)$, then $c \in SP_t(b)$.

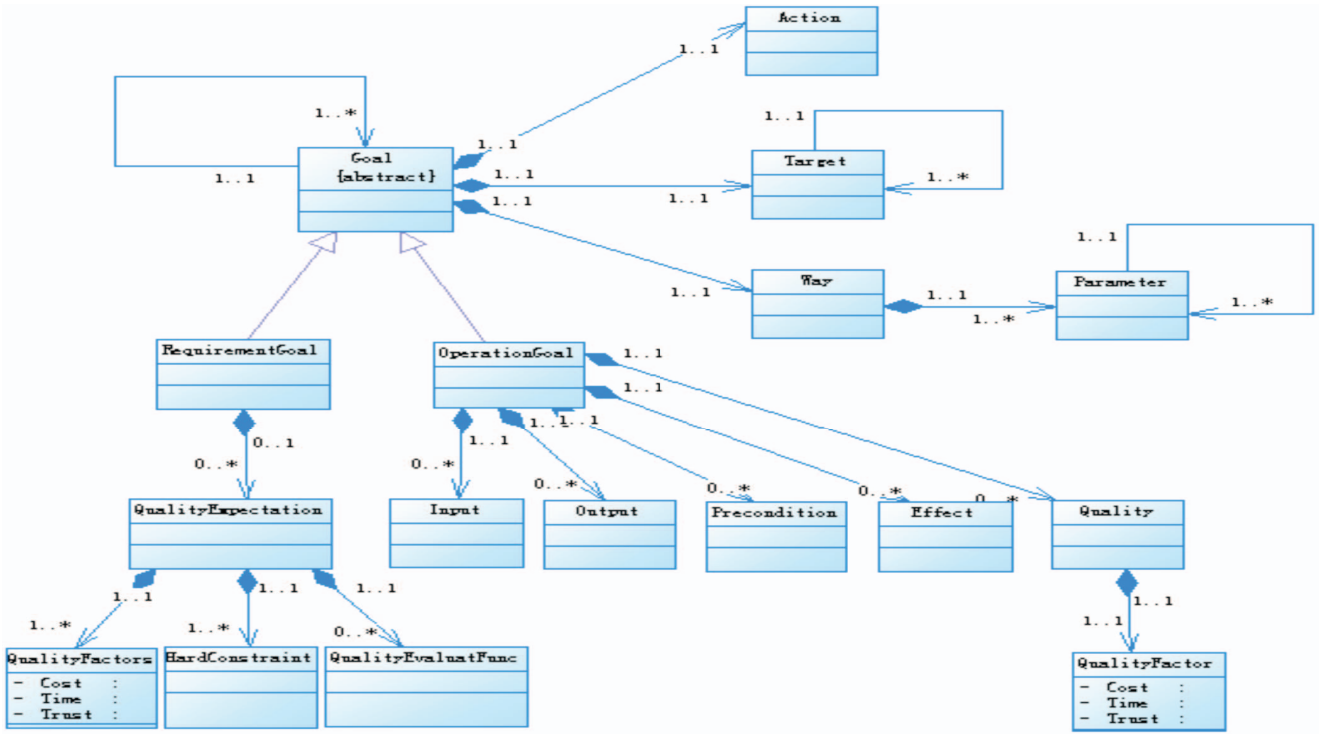

Figure 2. Goal Structure

For two sets A and B, if $\forall b_1 \in B$, $\exists a_1 \in A$, $a_1 = b_1$ or $a_1 \in SP_t(b_1)$, then A specializes B and it is denoted as $A \subseteq SP_s(B)$. If $A \neq B$, then denoted as $A \subset SP_s(B)$.

A. The Semantic Goal Model

When an outside request is received, an agent should decide the action it should take. This is often achieved through a goal deliberation process in a cognitive agent. There are many research works on goal modeling in recent years. In these works, goals can be roughly divided into two types, i.e., declarative goals and procedure goals. Declarative goals are about situations or states, and procedural goals are about actions. In the SIA structure, the concept of procedure goal is adopted.

In related work, a procedure goal is often described as a string. As a service broker, the SIA should be able to differentiate similar web services and organize them according to not only the functional requirements but also qualitative requirements. Current goal representation in agent is not enough for the SIA due to following reasons:

- Goal representation is too simple: Request coming from outside is often very complex and consists of multiple aspects. For example, a weather information query request can be based on zip code or city name;
- Quality information is not included in the current goal representations: A quality specification concerns how well the service offering might be. Quality requirement can be represented as an expectation on overall quality score. In addition, constraints over one or more quality factors are often specified.
- More relational types among goals are required: Generally, goals and their relationships can be described as a predefined AND/OR graph. In this graph, depending on the decomposition type, all (for AND) or at least one (for OR) of the sub-goals has to be implemented to satisfy their parent goal. The goal deliberation of the agent is a process traversing the goal graph to decompose an abstract goal into a set of sub-goals that can be satisfied by executing plans directly. Since new plans to integrate web services can be added into the SIA, the fixed goal graph is difficult to adapt to the ever-expanding web service integration plan set.

In order to extend the goal representations, the goal model of the SIA is defined based on ontology so that the generalization (or specialization) can be inferred according to their definitions.

The definition of a goal is shown in Fig. 2. The goal includes an action, a target and the way. Goals are divided into requirement goals and operational goals. Requirement goals are those an agent tries to satisfy. It includes quality expectations, which consists of quality factors, constraints and a quality evaluation function. The default quality factors include time, cost and reliability. An operation goal represents the function and quality that a plan can achieve. Its specification also has inputs, outputs, preconditions and effects.

There are two relationship types among goals:

1. Decomposition Relationship

A goal g can be decomposed into several sub-goals $G'=\{g_1, g_2, \ldots, g_m\}$ and each sub-goal $g_i=SubOf(g)$ (i=1, 2, …, m) will contribute to the partial fulfillment of g. For the goal g, a sub-goal g_i can be optional or indispensable.

- AND($g_1, g_2, \ldots, g_m$): In order to satisfy goal g, all goals should be satisfied.
- OR($g_1, g_2, \ldots, g_m$): In order to satisfy goal g, one of goals is selected to be satisfied.

2. Generalization (Specialization) Relationship

If goal g_1 can be satisfied by realizing another goal g_2, then we call g_2 specialize g_1 and denote it as $g_2 \in SP_g(g_1)$. Accordingly, we can call g_1 generalize g_2 and denote it as $g_1 \in GE_g(g_2)$.

If $verb(g_1)$ is different from $verb(g_2)$ then their specialization relationship should be defined explicitly. If $verb(g_1)=verb(g_2)$, then specialization relationship can be determined using the following reasoning mechanisms:

Suppose a goal g, where $verb(g)=v$ and $target(g)=t$ with parameter set $P=(p_1, p_2,..., p_n)$ and a goal g', where $verb(g')=v$ and $target(g)=t'$ with parameter set $P'=(p_1, p_2,..., p_n,..., p_m)$. If $t' \in SP_t(t)$ or $t'=t$, and for $\forall pv(p_i, g')$, $pv(p_i, g') \subseteq SP_s(pv(p_i, g))$, then g' specializes g. The specialization relationship among goals can be inferred from the specialization relationships of the corresponding action, object and way.

B. The Goal Deliberation Process

When a request is received, a goal deliberation process is started. This is triggered according to a request and goal association rule, which is denoted as $R \rightarrow GA$. R is a request type definition. GA can be a rule defining how to generate the requirement goal according to the contents of a request.

The goal generated by the GA can be a high-level one. Through the deliberation, a requirement goal can be decomposed to a set of sub-goals, for which a set of plans are defined to implement them. The goal generated by the GA can also be a low-level goal for which plans can be found to implement it directly.

IV. PLAN MODELS FOR WEB SERVICE INTEGRATION

When a request is received, the SIA tries to organize and execute plans in response to it. Therefore, on the one hand, information should be attached to a plan supporting the selecting process. On the other hand, a plan should define the way to organize services. The former one is called semantic model that is based on operational goal annotation, the later one is called syntax model, which can be represented as a set of logic flows and data flows among activities.

A. Goal Annotation for the Web Service Integration Plan

When a web service is integrated into the SIA as a plan, the operational goal should be annotated to this plan. It is a difficult task if it is done manually. We designed a semi-automatic framework to annotate the operational goal to the web service integration plan.

The WSDL file is a universally supported interface to describe web services. Consequently this framework takes the WSDL files as corpora to automatically generate operational goals. This process is composed of 4 steps:

1. WSDL Parsing

In this step, the *operations*, *inputs*, *outputs* and *schema types* are extracted.

2. Raw Information Processing

Because web service is a software component, operation and parameter naming often follow some naming convention in programming, for example, abbreviation or adding prefix. Therefore, we process the information extracted from WSDL in three steps: segmenting text according to the rules in terms of Pascal or Camel convention, striping the prefix or suffix, and expanding the abbreviations according the dictionary.

3. Natural Language Processing

In this step, information is extracted to fill the structure of goal so that an operational goal instance is created. Since the WSDL documents are used for interoperation so that they are always named in accordance with some patterns. Once the pattern is identified, the correct mapping rules can be applied to extract syntactical elements for semantic attributes.

It is feasible to identify pattern and extract semantic information from syntax tree. In order to match patterns in syntax tree, our automation framework adopts Tregex [17] which is a kind of tree pattern expression extended Tgrep2 [18]. A two-stage pattern based approach is designed to extract semantic information from the natural language text. The first stage is to discriminate the syntactic pattern it belongs to, and the second stage is to extract phrases from the discriminated syntax structure.

4. Ontology Alignment

After the above three steps, the structure of goal has been filled. But the phrases may not be standardlized. In order to find the most matched ontology for these words or phrases, SPARQL based ontology alignment is applied. ARQ [19] is an implementation for SPARQL standard, which provides two kinds of extension methods (a) Filter Function (b) Property Function. In our system, we mainly extended the property function to integrate the capabilities of string normalization, similarity algorithms, similarity measure strategy and match strategy into the SPARQL statements.

The above method extracts the information according to the goal structure. It needs to be justified, modified or completed. The initial quality information can be defined when a web service is added to the SIA. But it can also be updated according to the message or updated periodically according to the statistic information by itself.

B. The Syntactic Model for the Web Service Integration Plan

A web service integration plan model should define the logic and the sequence of the relevant data flow among service invocation activities.

A plan model is defined as $<S_t, RA>$, where S_t is the activity set; RA represents relationships among the activities. $RA=DF \cup LF$, where DF and LF are data flow set and the logic flow set respectively.

An activity can be in one of the following states: *waiting*, *ready*, *running*, *completed*, *overtime*, *failed* and *aborted*. When an activity changes its state from one to another, an atomic event happens. The content of each activity in the model can be specified as an internal service or a web service. When the state of an activity becomes *ready*, a service invocation goal *Initialize*(*Activity*("*A*")) is produced. In addition, the content of an activity can also be a requirement goal which is called abstract activity. Accordingly, the plan including one or more abstract activities is called abstract plan. Before the abstract plan can be executed, all abstract activities should be matched with plans through a goal deliberation process.

V. PLAN SELECTION

In generally, plan selection can be divided into two steps. The first step is to select the plans through the goal deliberation procedure. The second step is to decide which solution is the best according to quality requirements.

The quality of a plan can be calculated based on the quality information of each service in the plan.

Suppose A_i is an activity of a plan and it's time, cost and reliability are T_i, C_i and R_i respectively. Since a plan can be constructed through a set of standard blocks iteratively, we only need to figure out how to calculate the quality information of typical block types:

1. Sequence Block:

$$T_P = \sum_{i=1}^{n} T_i, C_P = \sum_{i=1}^{n} C_i, R_P = \prod_{i=1}^{n} R_i$$

2. Parallel Block:

$$T_P = Max(T_1, T_2 \cdots, T_n), C_P = \sum_{i=1}^{n} C_i, R_P = Min(R_1, R_2 \cdots, R_n)$$

3. Alternative Block:

(suppose the possibility of executing activity A_i is γ_i)

$$\sum_{i=1}^{n} \gamma_i = 1, T_P = \sum_{i=1}^{n} T_i * \gamma_i, C_P = \sum_{i=1}^{n} C_i * \gamma_i, R_P = \prod_{i=1}^{n} R_i * \gamma_i$$

For a requirement goal, the algorithm to select the best plan is as following:

Algorithm1:

```
chooseOptimalPlan(RequirementGoal rG, PlanSet PlanS)
{  //The candidate set of operational goals
   candidateSet=null;
   //Plan selection according to semantic information
   For each plan in PlanS{
      oG=operationGoal(plan);
      if oG∈SP_g(rG) add oG to candidateSet;}
   //Hard constraints checking
   parseRG_HardConstraint(rG);
   for each oG in candidateSet{
      if(misMatch(oG.quality, rG.hardConstraintFunc)){
            remove oG from candidateSet;}
   }
   //Select the best one according to the quality score
   parseRG_QualityEvaluationFunction(rG);
   MAX←-∞;
   planID←0;
   for each oG in candidateSet{
   //Fe is a quality score function
   If (Fe(oG.quality)>MAX){
         MAX←Fe(oG.quality);
         planID←getPlanID(oG);}
   }
    return planID;
}
```

In most cases, feasible plans cannot be found to achieve a requirement goal directly so that a deliberation process is started to generate feasible plan set.

The deliberation process will generate an AND/OR tree graph called *planGraph* in terms of the original goal graph and the plan set. The algorithm is as following:

Algorithm2:

Suppose *rG* is the root of the *planGraph* and *requirementGoalSet* is an empty set:

```
deliberationProcess(RequirementGoal rG)
{
   if rG is an AND-decomposed node in goal graph {
      append all sub-goals of rG to the sub-nodes of node(rG) of
      planGraph as AND-decomposed nodes;
      }
   if rG is XOR-decomposed node in goal graph {
      append all sub-goals of rG to the sub-nodes of node(rG) of
      planGraph as OR-decomposed nodes;
      }
   insert sub-goals of rG into requirementGoalSet;
   if rG can be achieved by executing a plan {
      select all feasible plans according to semantic information and
      append them as OR-decomposed sub-nodes of node(rG) of
      planGraph;
      remove rG from requirementGoalSet;
      }
   if requirementGoalSet is not empty {
      bring one requirement goal rG_1 from requirementGoalSet;
      deliberationProcess(rG_1);
}
```

Feasible solutions can be extracted from the *planGraph* easily: for a node selected, if it is AND-decomposed type, it's child nodes are all selected. If it is OR-decomposed type, one of it's child nodes is selected. A feasible solution consists of all selected bottommost plan nodes. We can regard the selected plans as the activities of a parallel structure and the quality can be calculated accordingly.

When an abstract plan is found, a *planGraph* also needs to be generated. In this *planGraph*, the root represents the abstract plan and a set of AND-decomposed type sub-nodes are appended. Each sub-node represents the requirement goal of an abstract activity. Furthermore, these requirement goals can be deliberated according to Algorithm2 so that the *planGraph* can also be expanded.

VI. CASE STUDY AND IMPLEMENTATION

A. Case Study

Suppose we want to develop a SIA for providing weather information service, which is named as *WeatherInformationService* Agent. In this agent, four web services are registered. Their operation goals are defined as following.

	Operation from WS_1	Operation from WS_2	Operation from WS_3	Operation from WS_3
Action	Query	Query	Query	Query
Target	Weather Info.	Weather Info.	Weather Cond.	Wind Info.
Way(Parameter)	Zip Code	Country	Country	Country
		City	City	City
Time(ms)	10	8	10	10
Cost	0	0	0	0
Reliability	0.9	0.85	0.9	0.9
Accuracy	8	8	9	9
Integrality	7	7	5	5

Besides default quality factors, two extra quality factors including accuracy and integrality are also defined. The accuracy means the precise of weather information, which is represented as an integer ranging from 0 to 10. The integrality means how many days it can predict and it is assigned an integer ranging from 1 to 7.

Four web service integration plans are defined in *WeatherInformationService* Agent. Plan 1 is defined for integrating the operation of WS_1 and it includes three activities, one for obtaining Zip Code from message, the second for invoking web service and the third for composing the return message (*configureReturnMessage* activity). The other three plans are similar. They all include two parallel activities, one for obtaining the country name (*setCountryNameValue* activity) and the other for obtaining city name (*setCityNameValue* activity) from the message. After these two activities, the web service operation is invoked. And the last activity is for composing return message. Their graphic models are shown in Fig. 3.

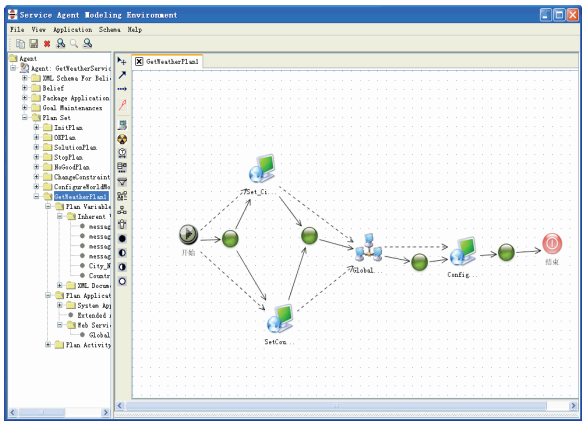

Figure 3. Plan Model for Web Service Integration

The semantic information can be added to the plan, for example, the semantic information of $plan_1$ is specified as following:

```
<Plan1>
  <OperationGoal>
    <Action>Query</Action>
    <Target>WeatherInformation</Target>
    <Way> <Parameter>Zip Code</Parameter></Way>
    <Quality>
      <Time value = "10"/>
      <Cost value = "0"/>
     <Reliability value = "0.9"/>
     <Accuracy value = "8"/>
     <Integrality value = "7"/>
    </Quality>
   </OperationGoal>
</Plan1>
```

In order to let the agent be able to process the weather information query request, a requirement goal generation rule is defined for this query message. The rule file is:

```
global edu.sjtu.grid.agent.modeling.model.goal.RequirementGoal rg;
rule "Parse_GetWeather_IncomingMessage_ByCCName"
   when
        event: MessageIncomingEvent (messageTemplate ==
"GetWeather" , inputParameters contains "City", inputParameters
contains "Country" , qualityExpectationMap:qualityExpectationMap,
agentID : agentID, eventID : eventID)
 then
        rg.setAction(new Action("Get"));
        rg.setTarget(new Target("Weather"));
        rg.getWay().getParameterList().add(new
Parameter("CityName"));
        rg.getWay().getParameterList().add(new
Parameter("CountryName"));
        rg.setQualityFactors(qualityExpectationMap);
        rg.saveRequirementGoalInDB(agentID, eventID);
```

Through a rule engine (in our case, Doorls is used as the rule engine), the message is translated into the requirement goal as following:

```
<RequirementGoal>
        <Action>Query</Action>
        <Target>WeatherInformation</Target>
        <Way>
                <Parameter>Country Name</Parameter>
                <Parameter>City Name</Parameter>
        </Way>
        <QualityFactors>
                <QualityFactor name = "Time"/>
                <QualityFactor name = "Cost"/>
                <QualityFactor name = "Reliability"/>
                <QualityFactor name = "Accuracy"/>
                <QualityFactor name = "Integrality"/>
        </QualityFactors>
        <HardConstraints>
                <Constraint>
                <Operator value = "LT"/>
                <Factor value = "Time"/>
                <Value value = "10"/>
                </Constraint>
                <Constraint>
                <Operator value = "E"/>
                <Factor value = "Cost"/>
                <Value value = "0"/>
                </Constraint>
                <Constraint>
                <Operator value = "EGT"/>
                <Factor value = "Reliability"/>
```

```
                <Value value = "0.9"/>
                </Constraint>
        </HardConstraints>
    <QualityEvaluationFunction>
            Time*(-0.2)+Reliability*0.3+
            Accuracy*0.25+Integrality*0.2
    </QualityEvaluationFunction>
</RequirementGoal>
```

According to the goal representation, either of $plan_1$ and $plan_2$ can achieve the requirement goal directly. According to the goal relationships, *Query*(*WeatherInformation*)=AND(*Query*(*WeatherConditions*), *Query*(*WindInformation*)), this requirement goal can also be achieved by executing $plan_3$ and $plan_4$. Fig. 6 is the *planGraph* constructed.

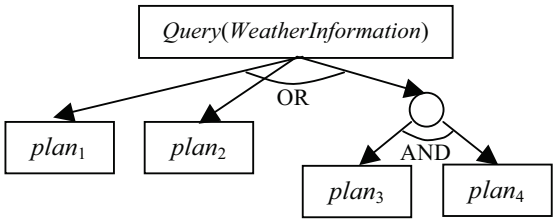

Figure 4. The *planGraph* for Query Weather Information

From the *planGraph*, we can obtain the solution set: {<$plan_1$>, <$plan_2$>, <$plan_3$, $plan_4$>}. Since $plan_2$ violates the quality constraint (Reliablity≥0.9), it is excluded.
The quality score of $plan_1$ is:
10*(-0.2)+0.9*0.3+8*0.25+7*0.2=1.67
The quality score of < $plan_3$, $plan_4$> is:
10*(-0.2)+0.9*0.3+9*0.25+5*0.2=1.52
Consequently, $plan_1$ is selected.

B. Implementation

Our implementation of SIA platform is based on Java. It consists of an agent modeling tool, an agent container and an agent directory.

Agent modeling tool is a graphical environment that allows user to build an agent in an interactive way. Fig. 5 is the tool to annotate the operational goal for a plan. Fig. 6 is the tool to define the data mapping relationship between world model and service operation's parameters for a plan. Agent specification is saved as an xml file. It can be deployed into containers through web services and at the same time it is registered into the directory.

Agent container is a distributed environment supporting agent running and communication. Currently, it is built on JADE environment.

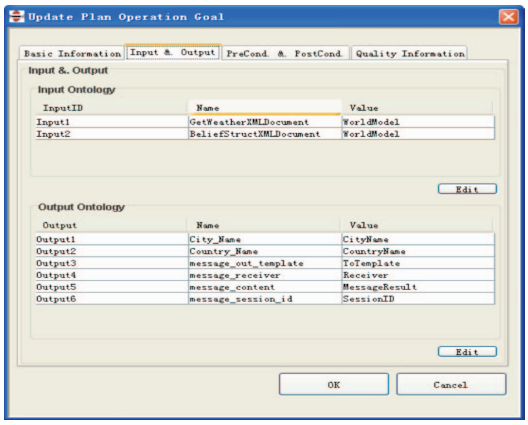

Figure 5. An Operational Goal Annotation Tool for Plan

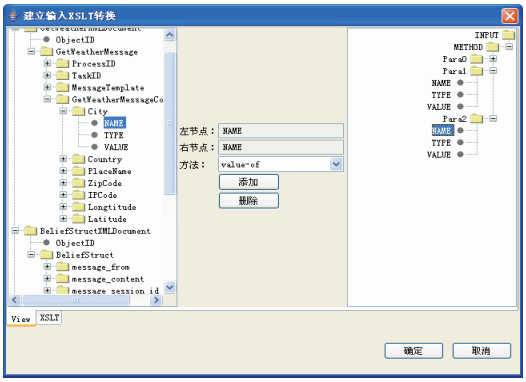

Figure 6. Data Mapping Relationship Modeling Tool for a Plan

VII. RELATED WORK

Because web services and agent technology can help each other, to integrate them together is becoming a trend.

In order to let the agent can call web services, some tools can generate a JADE proxy agent for an existing web service so that other agents can send ACL(Agent Communication Language) requests to the proxy agent [8][20]. In other works, the capabilities of an agent can be exposed as web services [10][11][20]. For example, the WSAG [20] manages the transition from agents to web services and the Generator is a support tool for generating agents that operate within the Gateway providing a concrete Web service interface for a particular external agent. All these efforts focus on connecting agents and web services. The service intermediary agent manages a group of web services and it can offer value-added services to the outside based on it's internal knowledge.

Some research groups also proposed the the agent model which can manage multiple web services. For example, in RACING framework, agent can mange several web services, and they can negotiate with each other to compose

web services [21]. These works try to make use of multi-agents to solve particular problems such as web service selection or web service composition. Comparing with these works, the service intermediary agent is a more general framework and it provides the general approach to integrate and manage web services. These features make SIA become a general framework which can produce different type service agents for a domain. These agents can be an important part of service collaboration infrastructure.

VIII. CONCLUSIONS AND FUTURE WORK

In order to improve the current web service technology, a framework called service intermediary agent is introduced in the paper. The SIA can provide value-added professional service based on management of a group of inherently related web services. A SIA can also extend and optimize its inherent knowledge to improve its service level autonomously. In addition, SIAs can cooperate with each other to satisfy the outside requirements jointly. We believe these features make service intermediary agents form important parts of the service collaboration infrastructure, which is very important to build an open service computing environment. This paper figured out the structure and working mechanism of the SIA and there are many research issues still waiting for investigating. The future work will focus on but not limit to:

- Design collaboration mechanism for service intermediary agents;
- Try to provide self-leaning algorithms to optimize the SIA's plan set;
- Explore how to incorporate web services into the SIA from outside automatically.

ACKNOWLEDGMENT

The paper is partially supported by China National Science Foundation (Granted Number: 61073021, 61272438), Research Projects of STCSM (Granted Number: 11511500102).

REFERENCES

[1] Booth, D., Haas, H., McCabe, F., et al, Web Services Architecture, http://www.w3.org/TR/ws-arch/, 2004.2

[2] W3C, SOAP Version 1.2, http://www.w3.org/TR/SOAP/, 2007.4

[3] Christensen, E., Curbera, F., Meredith, G. et al, Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, 2001.6

[4] Papazoglou, MP, Traverso, P., Dustdar, S., et al, Service-Oriented Computing: State of the Art and Research Challenges, IEEE Computer, 2007, 40(11): 38-45

[5] Cao, J., Wang, J., Law, K., et al, An Interactive Service Customization, Information and Software, 2006, 48(4): 280-296

[6] Russell, S J., Norvig, P., Artificial Intelligence: A Modern Approach (2nd Ed.), Prentice Hall, 2003

[7] Huhns, M., Singh, M., Burstein, M., et al, Research Directions for Service-oriented Multi agent Systems, IEEE Internet Computing, 2005, 9 (6): 52-58

[8] Liu, S., Küngas, P., Matskin, M., Agent-Based Web Service Composition with JADE and JXTA, In Proceedings of the 2006 International Conference on Semantic Web Services, Las Vegas, Nevada, USA, 2006.1.1: 110-116

[9] Ian Dickinson, Michael Wooldridge, Agents are not (just) Web Services: Considering BDI Agents and Web Services, http://www.hpl.hp.com/techreports/2005/HPL-2005-123.pdf, 2005.7

[10] JADE, http://jade.tilab.com/, 2010.1

[11] Greenwood, D., Calisti, M., Engineering Web Service- Agent Integration, In Proceedings of IEEE Systems, Cybernetics and Man Conference, Hague, Netherlands, 2004.1.1: 1918-1925

[12] Agentbuilder, http://www.agentbuilder.com/, 2004.6

[13] Rao, A. S., Georgeff, M. P., Modeling Rational Agents within a BDI-Architecture, In Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning, 1991: 473-484

[14] Gruber, T. R., A Translation Approach to Portable Ontology, Knowledge Acquisition, 1993, 5(2): 199-220

[15] Chakravarthy, S., D. Mishra, D., Snoop: An Expressive Event Specification Language for Active Databases, Data & Knowledge Engineering, 1994, 14(1):1-26

[16] Pietzuch, P. R., Shand, B., Bacon, J., A Framework for Event Composition in Distributed Systems, In Proceedings of 4th ACM/IFIP/USENIX International Conference on Middleware, Rio de Janeiro, Brazil, 2003.6: 62-82

[17] Levy, R., Andrew, G., Tregex and Tsurgeon: Tools for Querying and Manipulating Tree Data Structures, In Proceedings of 5th International Conference on Language Resources and Evaluation, Genoa, Italy, 2006.5.22-28, http://nlp.stanford.edu/software/tregex.shtml, 2006

[18] Douglas L. T. Rohde, TGrep2 User Manual version 1.15, http://tedlab.mit.edu/~dr/Tgrep2/tgrep2.pdf, 2005.10

[19] ARQ, http://jena.sourceforge.net/ARQ/, 2009

[20] Agentcities Task Force, Integrating Web Services into Agentcities Recommendation, http://www.agentcities.org/rec/00006/actf-rec-00006a.pdf, 2003

[21] Ermolayev V., Keberle N., Plaksin S., Towards Agent-Based Rational Service Composition: RACING Approach, In Proceeding of the International Conference on Web Services Europe, Erfurt, Germany, 2003: http://www.old.netobjectdays.org/pdf/03/papers/icws/28530167.pdf167-182