Multi-agent Test Environment for BPEL-based Web Service Composition

Wenli Dong
Institute of Software, Chinese Academy of Science
Beijing, China
Wenli@iscas.ac.cn

***Abstract*—The distributed structure of agents makes the test for large and complex Web Service composition possible. This paper discusses how to develop the multi-agent test environment for BPEL-based Web Service composition. The BPEL-based Web Service composition is modeled by HPN that can be easily referenced by test case generation and test evaluation. By representing agent in HPN, the agent can be dynamically bound at runtime. In order to implement the multi-agent test environment for BPEL-based Web Service composition, the ontology is analyzed and defined based on XML because of its flexibility, extensibility etc. This ontology is the basis of the agent communication and provides the terms for test case generation and test evaluation etc. The test case generation and test evaluation under this test environment are analyzed. The test process is provided to illustrate the inter-action between the agents under this multi-agent test environment to accomplish test task.**

Keywords*—c *Business Process Execution Language for Web Services, High-level Petri Net, Multi-agent test

I. INTRODUCTION

The interoperability between Web Services has been one of the most important research topics on the SOA field with mounting economic and technical challenges as growing complexity and increasing services. Business Process Execution Language for Web Services (BPEL) [1] allows specifying business processes and how they relate to Web Services.

Testing Web Service composition to gain confidence in its conformance to the desired function with expected QoS is a key problem certainly, because lack of trust will prevent Web Service from adopting. But, because of its properties, Web Service composition test is complex, difficult and time-consuming, which makes Web Service test the following challenges: 1) the components in Web Service composition are interactive with a diversity of information formats and execution platforms. Testing Web Service composition requires a flexible and composite software environment to host and/or integrate a diversity of tools for various platform and language. 2) The dynamics of the Web Service composition such as publishing, binding, discovery, and composition dynamically demands software tool can bridge the gap between dynamic test and static analysis. 3) The provider and requestor are geographically distributed, the components in Web Service composition can be dispersed in different computers, and this property requires such a test environment that can be extended.

Up to now, the researches on Web Service and agent focus on their integration: a Web Service should be able to invoke an agent service and vice versa. [2][3][4] introduce architectures which connect agents with Web Service, and treat a Web service as an agent having proper ontological description. [5][6][7] propose how to access Web Services through an agent-based service gateway, and how to access agent-based services through a Web Services-based service gateway. However, testing Web Service in agent software environment isn't involved in these researches, neither is Web Service composition test. The agent software application in Web Service isn't furthered to construct a test environment to make use of the agent software to offer effective way for testing Web Service composition.

To meet these requirements, based on semantic Web and HPN (High-level Petri Net), by allowing software agents to communicate and understand the information published on the Internet, a multi-agent test environment is proposed. First of all, the agent runtime environment provides a dynamic mechanism for service description, invocation, discovery, and composition. Secondly, the semantic vision is to allow communications from software agent to software agent. Under our test environment, various software agents decompose test task into small subtasks and carry out these tasks. They cooperate with each other to fulfill the whole test task. The multi-agent test environment proposed in this paper is composite and extended, and agents can dynamically join and leave the system to achieve the maximum flexibility and extendibility.

This paper is organized as follows. Section 2 gives the architecture of the multi-agent test environment. All the test agents are described and classified in this section. In section 3, the way for agent binding dynamically is introduced, which is implemented based on HPN. Section 4 is generation of test environment. Based on the lifecycle of test environment construction, the ontology of test environment, modeling of BPEL-based Web Service composition, test case generation, and test evaluation under the multi-agent test environment are discussed. The test process under this test environment is

* This work is supported by the National High Technology Research and Development Program of China (Grant No.2007AA01Z190)

illustrated in section 5. Section 6 is the conclusion and future work.

II. TEST ENVIRONMENT

As shown in Fig.1, the test environment consists of a number of agents to fulfill test tasks for BPEL-based Web Service composition. Organizing relative agents into group is convenient for controlling agents [8][9][10]. As shown in Fig.1, the Get BPEL (GB) agents and the BPEL Analysis (BA) agents are arranged in media agent group for their interaction with BPEL/WSDL specification and test environment. However, these agents can be distributed in different computers. Actually, the distribution of agents is free according to any specific configuration, can move and change their location at runtime.

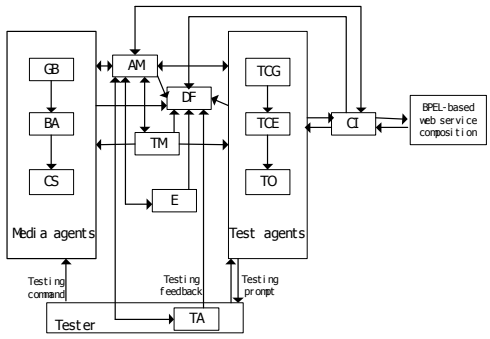

Fig.1. Agents for Testing BPEL-based Web Service Composition

Get BPEL (GB) agents obtain BPEL/WSDL specification from Test Assistant Agent. The BPEL and corresponding WSDL documents will be input by GB while Test Assistant Agents provide graph interface for user.

BPEL Analysis (BA) agents analyze the BPEL/WSDL specification, exact useful information, and construct HPNs for basic activity on a HPN platform. The structure information of basic activity in HPN form is stored in Knowledge Base (KB), a database that is used to store information related with test.

Composition Structure (CS) agents analyze the structure of BPEL-based Web Service composition, and generate a HPN presentation to describe the structure.

Test Case Generator (TCG) agents generate test cases to test an activity according to certain test criteria.

Test Case Execution (TCE) agents execute the test cases, and generate execution results. Two ways of test case execution can be adopted in our design. One is to run the test cases interactively under the control of Agent Manager (AM), with the aid of a Test Assistant (TA). The other is to playback.

Coordinate Interface (CI) agents provide flexibility for every kind of Web Service implementation. CI plays the role of test harness, test driver and module stubs. They enable the integration of various test tools seamlessly into the multi-agent test system so that components written in different languages can be tested in a uniform environment.

Test Oracles (TO) agents verify whether the test results match the given BPEL/WSDL. BPEL/WSDL Specification describes constraints on the order in which events can occur in executions of an agent system and constrains for message, operation etc. Oracles produced from these specifications can verify whether or not traces generated by executing a system conform to the specifications.

Test Assistance (TA) agents provide the interface between tester and computer that guides testers in the process of test. For example, the BPEL/WSDL documents are input from TA to TCG to generate test cases. The test result, test log and test report can be obtained by TA.

Test Monitor (TM) agents monitor the test process at runtime and store the monitoring information in KB for later analysis.

Evaluation (E) agents are used to collect monitoring information and log information, employ predefined evaluation model, so that appropriate conclusions about the quality of the Web Service composition can be drawn. The rank information based on evaluation result is recorded and referenced by later test.

Agent Management (AM) agents are agents in charge of managing agents, including agent description, agent arrangement. In some degree, it is corresponding to ASM (Agent Management System) [11]. An agent management agent has the following capabilities:

1) Creating an agent description. It is based on XML. The description consists of agent name, agent locator, and agent properties. Among above properties, agent name is unique generated by combining owner to identify different agents for later agent selection. E.g., for an agent in computer called tt owned by IBM, and implement add function, its name can be represented as IBM::tt::addAgent1 simply. The agent ontology is introduced in section 4.1.

2) Processing a given query for agents. The query condition can be composite to find suitable agent exactly. The query language is XQuery-based [12] because of its extensibility and flexibility. For example, a query for an agent can have the following structure:

parent::operation[attribute::name="deposit"]

The above expression is to select all parents of the context node that are elements named "operation" and whose name attribute has the value "deposit".

Directory Facilitator (DF) agent is mainly responsible for registering agent to make agent in a test environment visible to AM. Its register function is similar with UDDI (Universal Description, Discovery and Integration). But, DF pays more attention to QoS of service while UDDI focuses on the location of service.

The message mechanism consists of a set of communication primitives such as send, receive message that is passed between agents. Its design objects are applicable, flexible, lightweight, and simple.

In our prototype system, the communication mechanism is based on Message Queue. Two (or more) processes can exchange information via message queue. Via some message treatment module, the sending process places a message in a queue that can be read by another process. Each message is

given an identification and type so that processes can select the appropriate message. Process must share a common key in order to access to the queue in the first place. Message queue consists of queue manager, channel and queue. The two kinds of channel is sending channel and receiving channel. The queue is classified into transporting queue, local queue, and remote queue based on the message source or/and destination. The operation for message queue consists of control operations e.g. initialing message queue, sending/receiving operation etc. As a loose couple distributed communication way, message queue is independent of hardware and operation system, and can ensure data not being lost and copied. It provides an effective communication mechanism for agent.

III. REPRESENTING AGENTS IN HPN

Referencing the soft gene definition stated in [13]: a soft gene is an entity consisting of a set of behaviors and attributes. In a HPN, a behavior can be represented by a transition and the attributes can be represented by predicate properties. We define the behavior and attributes:

```
// predicate definition
struct pred { //attribute definition}
 ram <pred> predicate-name
//transition definition
trans name {  //  declarations
                  // arcs with the fire rule of transition
                 action { // code to evaluate at fire start. }   }
```

Under multi-agent test environment, the agent takes part in test or not is decided at runtime. AM agents arrange the work for all available agents, and the state of agent is recorded by AM agents. Under thus open and dynamic environment, agents change state dynamically and interact with other agents. An agent can be defined as an entity with a set of soft gene. At a specified time, the steps of the agent taking are decided by its state and properties at this time. Based on above analysis, we can represent agents in HPN. The agent name is defined in predicate properties that denote the agent to take charge of transition, and bind/discard agent name with a concrete agent is dynamically. AM agents organize the interactions of all available agents.

Assuming P is the set of predicates; A is the set of agents; bind represents the relationship between a predicate and an agent. That is:

$A = \{a_0, a_1, ..., a_n\} \quad (n \geq 0)$

$P = \{p_0, p_1, ..., P_m\} \quad (m \geq 0)$

bind: $A \times P \rightarrow \{0,1\}$

The value of bind is 1 means at predicate P_x, agent a_y is in the state specified by P_x.

Two types of operations are added in a HPN platform. One is bind (P_x, a_y), that means building relation between P_x and a_y. The other is release (P_x, a_y), that means discarding the relationship between P_x and a_y.

For each agent a_k, there is a set of behaviors represented by transition and pointed by arcs. Every arc will be labeled with the predicate information including agent name, input etc. the behaviors for agent can be represented as $T_s = \{t_0, t_1, ..., t_o\}$, and the arcs that fire each transition t_q can be represented as $ARC_{t_q} = \{arc_0, arc_1, ..., arc_w\}$. A simple sketch is shown in Fig.2.

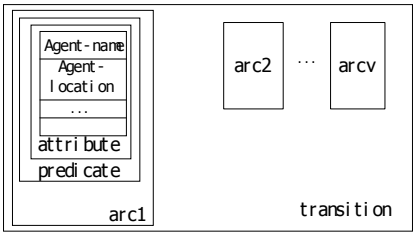

Fig.2. Binding Agents in HPN

IV. GENERATING MULTI-AGENT TEST ENVIRONMENT

In order to test composition dynamically, the test environment need start, stop, suspend or resume the operations of the cooperative agents. For analysis purpose, monitoring data, log, execution result are collected and manipulated, and will be used in evaluation of the tested Web Service composition. This means that the test environment should be able to model and control the lifecycle of agents. That is, the test environment is based on lifecycle. This ranges from ontology definition for providing the interaction between agents and BPEL specification, modeling the BPEL-based Web Service composition to arrange the test steps; right up to run the agent test, record the result, analyze and evaluate the test result, log, and monitoring data.

A. Ontology Definition

Ontology is an explicit specification of some topic. Ontology defines the basic terms and relations comprising the vocabulary of the topic and provides the rules to define the combination of the terms and the relations. Here, it consists of terms relative with multi-agent test for BPEL-based Web Service composition and the relations. Thus, the ontology of multi-agent for testing BPEL-based Web Service can be classified into two types: for multi-agent and for BPEL-based Web Service.

Referencing the ontology definition in [9][11], combing with the characteristics of BPEL-based Web Service composition, the simple ontology of multi-agent for testing BPEL-based Web Service is proposed in this section.

Generally, ontology modeling includes the Knowledge Interchange Format, UML, and XML. Because XML has the advantage of extensibility, flexibility, and readability [14], in this paper, we use XML Schema to define the ontology.

The Schema description is constructed based on the description components and their relations. Because of space limit, here, the agent description Schema will be given. For

message, transport, action, test case, test execution, test log/monitor, and test evaluation, the description Schema will not be provided.

Agent description includes: 1) Agent attribute: A set of properties associated with an agent by inclusion in its agent-directory-entry; 2) Agent communication language: A language with a precisely defined syntax semantics and pragmatics, which is the basis of communication between independently designed and developed agents; 3) Agent directory entry: A composite entity containing the name, locator, and attributes of an agent; 4) Agent locator: An agent locator consists of the set of transport descriptions used to communicate with an agent; 5) Agent name: An opaque, non-forgeable token that uniquely identifies an agent.

An agent can be defined as:

<element name="Agent_name" type="xsd:string"/> <element name="Agent_locator"><complexType><sequence> <element ref=" Transport "/></sequence> </complexType></element><element name="Agent_communication_language" type="xsd:string"/><element name="Agent_attribute"> <compexType><sequence><element name="a_attribute" type="xsd:string"/></sequence></complexType> </element> <element name="Agent_directory_entry"> <complexType> <sequence> <element ref="Agent_name" minOccurs="1" /> <element ref="Agent_locator" minOccurs="1"/> <element ref="Agent_attribute" minOccurs="0"/></sequence> </complexType></element><element name="Agent"><complexType><sequence><element ref="Agent_directory_entry" minOccurs="1" /><element ref="Agent_communication_language" minOccurs="1" /></sequence></complexType></element>

B. *Modeling BPEL-based Web Service Composition*

We use HPN to model BPEL-based Web Service composition [15]. From the operation mapping to the service composition mapping for the various controlling constructs specified by BPEL/WSDL specification, the relationship between BPEL/WSDL conceptions and HPNs is specified in four levels according to intra-activity, inter-activity, intra-service, and inter-service.

In HPN, in order to map an operation, a part will be presented by a place with token whose type specified by the part type used as the interface of test case generation. An arc is used to link the transition with another arc linked to the input/output message consisting of parts. The physical preconditions described in BPEL are embodied in the HPNs by places. And the cause-effect analysis is adopted in this mapping.

The operation cluster is generated by basic activity such as receive, reply, assign, invoke, empty, terminate, and wait. The operations in an operation cluster have message interaction associated by operation dependency. We can make a corresponding action or statement to a transition. Places connected to the transition intuitively to express the states before and after executing the corresponding action or statement. Firing a transition means that the corresponding action is being executed. Operation invocation can be expressed by entering a token in a place that denotes the starting point of the operation.

The operation invocation sequences at service level can be loop, choice, link, parallel, and sequence, presented respectively by structured activities: while, pick/switch, link, flow, and sequence activity in BPEL. The transition of activities at service level implies other problems, i.e., dynamic binding and concurrency. Thus we should compute which condition is satisfied and select an operation that should be activated when a composite activity occurs. The mechanism of this computation and selection can be implemented by a HPN. To compute which action will be invoked, we attach the information of the action name to a token and the condition judgment on arc based on global parameters and token value to denote the action selection. Identifying information will be attached to token in places which can be implemented in HPNs. Selecting a operation to be invoked is done with the evaluation of "Guards" attached to arcs denoting operation invocations.

In Web Service composition, an operation in one service may have the same name with another one in another service. In test of BPEL-based Web Service composition using HPNs, this problem can be resolved by presenting the attribute value of a corresponding token by combining with the namespace of the invoked service for identifying the corresponding operation.

C. *Test Case Generation*

The problem of the amount of computation costs for concurrent and dynamic BPEL-based of Web Service composition can be resolved by using HPN. The reason is as following: Since a HPN is an FDT (Fuzzy Decision Tree) that models concurrent systems, and constrains that are involved in the BPEL based Web Service compositions. Thus, through the generation of a reachability tree using colored tokens, no unreachable states are generated.

However, when modeling a large system in a traditional Petri net, many equivalent subnets are often involved. HPN can resolve this problem, since HPN allows tokens having attributes, and hierarchical design is possible.

Reduction by the partition equivalent markings is used in the test case generation so as to obtain a test suite with reasonable length. The essential nature of the equivalent markings fits the basic idea of the teat case generation method "equivalence partitioning" [16] well. The idea is that one test case of an equivalence class has the same detecting error ability with other test case in the same equivalent class.

In this section, the STPC (Scenario Tree Based on Path Coverage) method will be introduced. The test suites generation procedure is: 1) A specification of the BPEL-based Web Service composition system described by a HPN platform is provided; 2) A STPC is constructed from this specification, and a test suite is generated from the STPC; 3) A test suite is a set of input sequences and correct output sequences. The input sequences and the output sequences are generated from traces of arcs, and traces of nodes from the root to leaves of the STPC respectively.

The test suite generation method for STPC is: 1) N is a finite set of nodes of STPC after reduction; 2) A is a finite set of arcs of the STPC; 3) Assuming there are k+1 leaf nodes $l_o,\ldots,l_k$, there are traces from the root node to every leaf node, if there are m nodes on a path to leaf $l_i(l_i = n_{mi})$, the finite sequence

of nodes and arcs on a trace of the leaf l_i can be defined as: $X_i = n_r \bullet a_{01} \bullet n_{01} \ldots a_{mi} \bullet l_i$

Where n_r is the root node, $n_{ji} \in N, a_{ji} \in A$, and " $\bullet$ " is a concatenation.

Then the test suite Π is: $\Pi = \{X_o, X_1, \ldots X_k\}$

Actually, a test suite is generated based on Depth-first Traversal, and so it can cover all paths.

The Petri tree for "five dining philosophers" is illustrated in Fig.3.

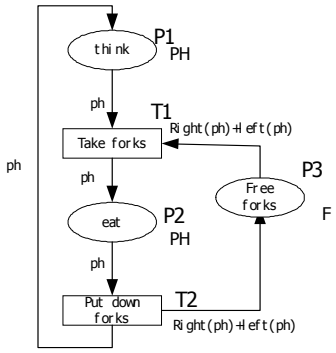

Fig.3. HPN for Philosopher System

It is reduced by covering markings (none in this tree) and by equivalent markings. The corresponding test suite: {{#1, #2, #3}}.

D. Test Evaluation

The quality of BPEL-based Web Service composition will be affected mainly by two reasons. One is the occurrence rate of the child components, actions, sub-scenario within a structured activity, and the other is the quality of child components. The multiple of them determines the contribution of the child components to the composition evaluation. While the quality of basic activity is presented by the value of token typed unsigned long int, the occurrence rate of the branch is attached to corresponding arc in HPN. The quality can be automatic calculated base on the HPN tool.

The quality of Web Service composition should be computed according to the quantitative metrics, e.g. reliability, every basic activity has a corresponding multi-dimension evaluation value to indicate its quality. Based on an adaptive evaluation method [17], basic activity quality will be computed quantitatively. Following the Web Service composition evaluation way, the quality of a BPEL-based Web Service composition can be calculated recursively.

For each structured activity in BPEL, based on their structure, we summarize following relationship between the structured activity and its sub activity.

The relationship between sequence/link activity and its sub activities is " $\cap$ ". According to probabilistic computation formula, assuming that a sequence/link activity, A, consists of B and C, the quality of B is B_{rel}, the quality of C is C_{rel}, B and C is independent. The quality of $A_{rel} = (B \cap C)_{rel} = B_{rel} \times C_{rel}$, this formula can be populated to sequence activity consisting of more sub activities.

The quality of while follows the formula: $A_{rel} = (B_{rel} \times C_{rel})^t$, where A_{rel} represents the quality of overall while scenario, C_{rel} represents relative occurrence rate of condition set by true, B_{rel} represents the quality of activity in while cycle, and t represents the times of loop.

The quality of pick can be formulated: $A_{rel} = C_{f1} \times B_{rel1} + C_{f2} \times B_{rel2} + \ldots + C_{fn} \times B_{reln}$, where A_{rel} represents the quality of overall pick scenario, C_{fk} represents relative occurrence rate of the k^{th} condition set by true, B_{relk} represents the quality of the k^{th} activity in pick.

The quality of switch can be formulated: $A_{rel} = C_{f1} \times B_{rel1} + (1 - C_{f1}) \times C_{f1} \times B_{rel2} + \ldots + (1 - C_{f1} - C_{f2} - \ldots - C_{f(n-1)}) \times C_{fn} \times B_{reln}$, where A_{rel} represents the quality of overall switch scenario, C_{fk} represents relative occurrence rate of the k^{th} condition set by true, B_{relk} represents the quality of the k^{th} activity in switch.

The quality of flow can be formulated: $A_{rel} = 1 - \prod_{k=1}^{n} (1 - B_{relk})$, where A_{rel} represents the quality of overall flow scenario, B_{relk} represents the quality of the k^{th} activity in flow.

For a ATM process described in Fig.4 simply, assuming the quality of each activity in ATM process has been evaluated at the pre-phase, and the result is shown in table 1.

Table 1. The Quality of Activity

Activity	Quality
connect	0.8
status	0.9
logon	0.7
withdraw	0.6
deposit	0.6
logoff	0.7
disconnect	0.8

A decision graph (Fig.4) illustrates the calculating process of the quality of the ATM process. Note that the value attached on the edge denotes the occurrence rate of the branch.

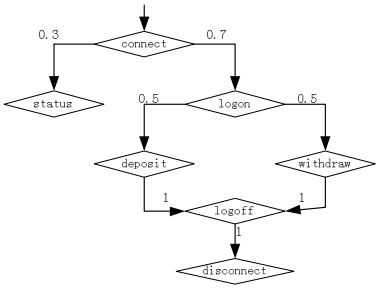

Fig.4. HPN with Decision Making

The quality of the ATM process:

$ATM_{rel} = 0.8 \times 0.7 \times (0.6 \times 0.5 + 0.6 \times 0.5) \times (0.7 \times 0.7 + 0.3 \times 0.9) \times 0.8$ =0.2043

V. TEST PROCESS

The multi-agent test environment for BPEL-based Web Service composition is a distributed system, involving various

agents and their interaction. Each agent can be identified by its unique name, and will be bound dynamically at runtime. The whole process is dependant on HPN assisting dynamically structure analysis. In this section, we will demonstrate the interaction process of the agents in the test process:

1) At the beginning of the test, all the agents will be created. Next, the agent description will be recorded by AM. The instantiated agents are registered in the DF, the corresponding rank information is set as 0 at this stage.

2) Under the control of AM, based on selection strategy, TA is selected by querying DF, so does GB, BA, CS, TCG, CI, TO, and TM. The BPEL and WSDL documents are input by GB through TA, and stored in KB. The structure specified by BPEL/WSDL is analyzed by BA and CS, and the HPN is generated. And TM will start its monitoring for all agent subtasks and record log in KB.

3) Based on STPC, the TCG will generate test suites and stores them in KB.

4) According to test composition, AM will send query request to DF, DF will find a suitable TCE and reply the request with the agent information including agent name, agent location etc. The specified TCE will interact with CI to transform the test script/program and TCE will execute test steps, all the test process will be recorded in test log.

5) The test result will be compared with the specified Web Service composition by TO and the matching result will be stored in KB.

6) To find next agent for next activity of BPEL-based Web Service composition test, the TCE completing test task will send message to indicate the finish of the pre-step to ask the AM to decide which agent can do the next test step.

7) The test executive sequence is controlled by AM after it stored the structure information in the form of HPN. Thus, the step 4, 5 and 6 will be repeated for all remaining activities with the agent querying to DF by AM.

8) When AM finds the test has been finished, the AM will send query request to find a suitable Evaluation agent to draw a conclusion for the quality of the tested Web Service composition.

Throughout the whole process, the agent will be ranked according to its behavior. The evaluation standard includes security, reliability, and performance etc [18]. The status of all agents will be recorded by AM dynamically.

VI. Conclusion and Future Work

In this paper, we have discussed how to develop the multi-agent test environment for BPEL-based Web Service composition. By representing agent in HPN, the agent can be dynamically bound at runtime. It is easily to re-compose the Web Service and select the suitable agents to carry out the test task with the help of HPN.

The distributed structure of agents makes the test for large and complex Web Service composition possible. All agents are coordinated by AM, and AM arranges the test tasks based on HPN representation of the Web Service composition. In order to implement the multi-agent test environment for BPEL-based Web Service composition, the ontology is analyzed and defined based on XML because of its flexibility, extensibility etc. The test case generation and test evaluation in this test environment are analyzed. The test process is provided to illustrate the interaction between the agents under this multi-agent test environment to accomplish test task.

Our future work includes perfecting the ontology of this test environment combing the OWL-S and the complex application of this multi-agent test environment to verify the reliability of the test environment. At the same time, the security of this multi-agent test environment is also within our further researches because of its distribution.

References

[1] Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[2] Jan Peters. Integration of Mobile Agents and Web Services. YR-SOC, Leicester, UK, 2005, pp. 53-58.

[3] J. Dale, L. Ceccaroni, Y. Zou, and A. Agam. Implementing Agent-based Web Services. AAMAS 2003, Melbourne, Australia, 2003.

[4] V. Ermolayev, N. Keberle, and S. Plaksin. Towards a Framework for Agent-Enabled Semantic Web Service Composition. International Journal of Web Services Research, Vol. 1, No. 5, 2004, pp. 63-87.

[5] Dominic Greenwood and Monique Calisti. An Automatic, Bi-Directional Service Integration Gateway. AAMAS, New York, USA, July 2004.

[6] Dominic Greenwood and Monique Calisti. Engineering Web Service - Agent Integration. Systems, Man and Cybernetics, Hague, Netherlands, October 2004, pp.1918-1925.

[7] J. Dale, A. H. M. Hajnal, M. Kemland, and L. Z.Varga. Integrating Web Services into Agentcities Recommendation. http://www.agentcities.org/rec/00006/actf-rec-0006a.pdf (Accessed 2005-02-09).

[8] LIU Ying-qiao, ZHAO Zheng-de et al. Web Service Based Multi-agent Cooperative Platform. Computer Engineering and Design, Vol.39 No.21, 2003, pp. 1269-1271.

[9] Qingning Huo, Hong Zhu and Sue Greenwood. A Multi-Agent Software Environment for Testing Web-based Applications. COMPSAC, Dallas, USA, November 2003, pp.210-215.

[10] Cecile Aberg, Patrick Lambrix, Nahid Shahmehri. An Agent-based Framework for Integrating Workflows and Web Services. WETICE, Linköping, Sweden, June 2005, pp. 27-32.

[11] FIPA Abstract Architecture Specification. http://www.fipa.org/.

[12] XQuery 1.0: An XML Query Language W3C Candidate Recommendation, November 2005, http://www.w3.org/TR/xquery/.

[13] Q. Yan, X. Mao, L. Shan, Z. Qi, and H. Zhu. Soft Gene, Role, Agent: MABS Learns from Sociology. IEEE/WIC, Halifax, Canada, October 2003, pp. 450-453.

[14] DONG Wenli, MENG Luoming. tML Schema Based ICS Proforma and Generation Method. Chinese Journal of Electronics, 2005(4), pp. 681-685.

[15] Wen-Li Dong, Hang YU, Yu-Bing Zhang. Testing BPEL-based Web Service Composition Using High-level Petri Nets. 10th IEEE International Enterprise Distributed Object Computing Conference, HongKong, October 2006, pp.441-444.

[16] G.J.Myers. The Art of Software Testing. John Wiiley & Sons, Inc., 1979.

[17] Wenli Dong. Research on Service Oriented Software Test Theory and Practice. Postdoctor Research Report, Tsinghua University Computer Science and Technology Department, China, 2006.

[18] Hang Ling, Ma Fanyuan. The Application of Web-based Software Agent in E-commerce. Computer Engineering, Vol.26 Supplementary, October 2000, pp.501