A Policy-based Adaptive Web Services Security Framework

Bin Li, Lingjun Zhao, Junwu Zhu, Jun Wu
School of Information Engineering
Yangzhou University
Yangzhou, China
Email: lb@yzu.edu.cn, zljbox@gmail.com, jdkr@163.com, j_wu@vip.sohu.net

***Abstract*—Web services security has become a hot topic in the research of service oriented computing. This paper aims to study many pivotal technologies in the web services security. Firstly, a policy-based framework for adaptive web services security is proposed, with the policy concept, management mechanism and execution mechanism can be separated effectively, moreover, by management of user context and web services context, web services access control can adapt to the changed environment. Secondly, a policy description language called ReiT is given, ReiT is a declarative language based on the rules and ontology and can express the structural and non-structural knowledge. A mixed reasoning mechanism is proposed, the web service access control policy including the user context and web services context can be evaluated by the reasoner. Finally, a policy aware BDI agent to authorize the access control of the web services is presented, and a prototype system based on Java EE and Jade Agent platform is implemented, Simulation experimental results and an example demonstrate the security framework is feasible and effective.**

***Keywords*—Web Service Security, Policy, Context-awareness, Ontology, Agent**

I. INTRODUCTION

Web services, which are issued in recent years, are a newly web-oriented development and integration framework for distributed applications. Web services represent more loosely-coupled distributed application architecture. And they provide an effective way for application to cooperate in an open environment. To guarantee the web services normal operation, the access control of the web services is employed. Therefore, research on the security framework of the web services is imperative[1][2].

Up to now, there are a number of access control models that have been developed, such as Role-Based Access Control (RBAC)[3], Task-Based Access Control (TBAC)[4], Provision-Based Access Control (PBAC)[5], etc.

Unfortunately, the current security technologies of web services have three limits: one is the lack of flexible, extensible, and loosely-coupled. The other is the limit of the network adaptability and could not adjust according to the dynamic environment. As in open systems, web services environment and their own states are ever changing that need to deal with the dynamic characters such as states and behaviors. Three is the lack of the autonomous. Web services themselves have no ability of proactive interaction. And they can not adjust their own states and behaviors.

To address the above mentioned problems, according to the features of web services, we propose a policy-based adaptive web services security framework (PAWSSF) in terms of its authorization architecture and policy formulation. By defining high-level rules, it can control and adjust web services behaviors without terminating their execution and without modifying the related execution mechanism. Moreover, this framework includes context[6] of user and web services that makes access control adapt to the changed environment.

The rest of this paper is organized as follows. Sect 2 introduces the logical architecture of PAWSSF. The management of context information is given in Sect 3. Sect 4 presents the policy description language ReiT and a mixed reasoning mechanism of rules and ontology. Sect 5 introduces a policy aware BDI agent framework. The prototype system of PAWSSF is implemented in Sect 6. Sect 7 is a case study to describe the usage of ReiT to define web services access control policy. Sect 8 is related work. Sect 9 is the conclusion and future work.

II. A POLICY-BASED ADAPTIVE WEB SERVICES SECURITY FRAMEWORK

In this section we describe how web services access control decisions can be achieved in the PAWSSF which is developed based on the IETF[7] policy framework. The architecture of PAWSSF is shown in Figure 1.

The main modules of PAWSSF are as follows:

① Context management: The user context and web services context are stored respectively in user context repository and web services context repository.

② Policy management: Web services policies described by ReiT are stored in policy repository. Administrator can modify the policies using the policy admin tool. The policies are evaluated by the policy reasoner which is based on the rules and ontology[8].

③ Agent management: There are three types of agents[9]: (ⅰ) User agent, (ⅱ) Web service agent: It acts

Project number: 60903130, BK2007074, BK2009698, BK2009699.
Corresponding author: Bin Li, Yangzhou University, Yangzhou, China. Email: lb@yzu.edu.cn.

doi:10.4304/jsw.6.12.2456-2463

as the Policy Enforcement Point (PEP), (iii) Policy decision agent: It acts as the Policy Decision Point (PDP). With the context of user and web service, the policy decision agent can call the ReiT reasoner to evaluate the web service access policy which are fetched from a policy repository.

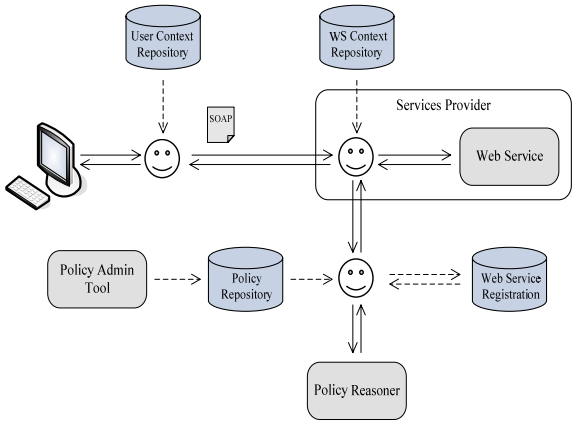

Figure 2. Policy-based adaptive web services security framework

The sequence diagram of successfully invoking the web service is shown in Fig.2.

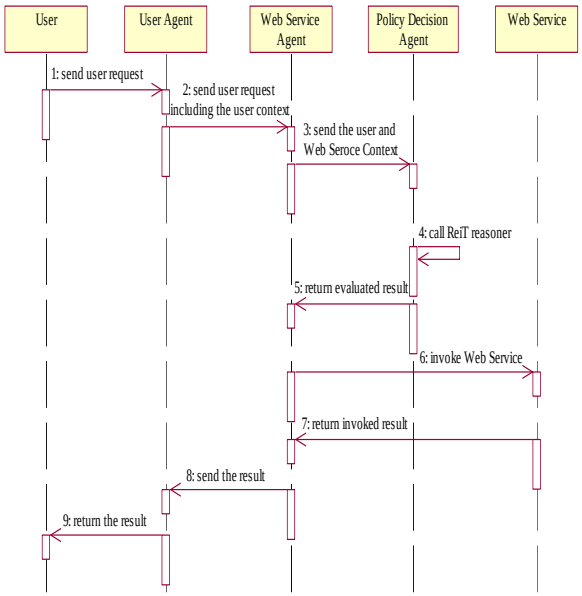

Figure 2. The sequence diagram of successfully invoking the Web Service

III. Context Information Management

Users and web services context is constantly changing in an open environment. To make access control decisions dynamically adapt to the environment change, PAWSSF should sense the context information of users and web services real-time, collect and manage the dynamic information of users and web services.

A. Context information ontology

For access control purposes, there are two types of context in PAWSSF:

① User context: user agent collects associated context which define the characteristics of the user, such as identifier, name, organization, role, etc.

② Web services context: web services have context that can play a part in making access control decisions, and a variety of web service context may be relevant to access control purposes, such as ownership, service taxonomy, state, QoS, etc.

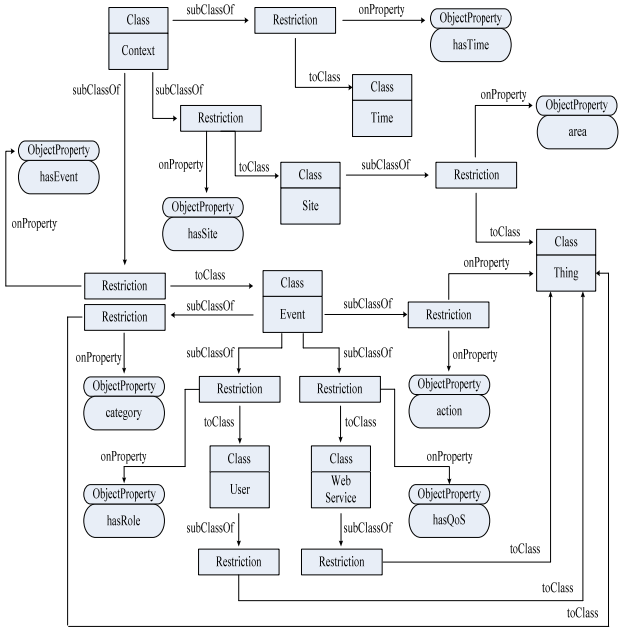

Figure 4. part of Context information ontology

B. Context information encapsulation

In PAWSSF, use agent, web service agent and policy decision agent interaction each to transmit context information. As the transmission process of context information ontology consumes a lot of network resources, the context information ontology is stored centralized and only the context information associated with policy is transmitted. Also, to take into account compatibility with the existing platforms, the context information of users and web services is encapsulated with SOAP message and FIPA ACL message respectively. Use agent and web service agent interact with SOAP, web service agent and policy decision agent interact with FIPA ACL.

① Context information encapsulation with SOAP message

In PAWSSF, context information transmit as a header block of SOAP message, context information header block is composed of several context information blocks, each of which is associated with a sort of context information. Each type of the context formation appears only once. As shown in Fig.4:

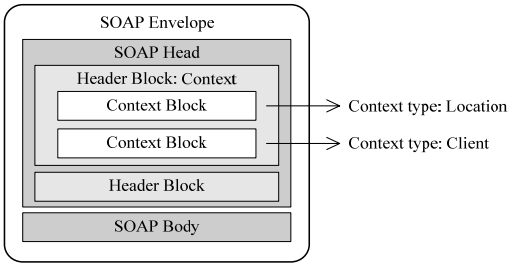

Figure 4. SOAP message with context information

In Fig.4, the context information header block contains two context information blocks, one is corresponding to Location type (location of user currently), and the other is Client type (the software and hardware information of user).

② Context information encapsulation with FIPA ACL message

In PAWSSF, context information stored in the content of FIPA ACL message is the same as SOAP message, the part of context information is composed of several context information blocks, each block associated with one sort of context information, as shown in Fig.5:

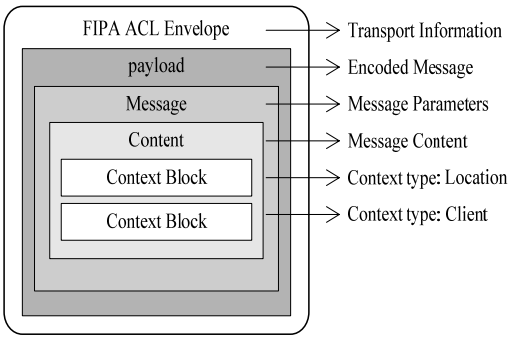

Figure 5. FIPA ACL message with context information

IV. POLICY MANAGEMENT

A. Policy description language ReiT

In PAWSSF, the ReiT, which can be based on any combination of users and web services context, is semantically richer and more expressive. The ReiT policy is based on rules and ontology. It is mainly represented in the form of rules which is based on the Rei[10]. Since the temporal has an important place in open environment, we add the temporal concept which acts as a general element. Moreover the temporal concept is represented by ontology.

Rules component of ReiT

The rules of ReiT mainly include two components:

① Basic Policy: Right, Obligation, Prohibition and Dispensation;

② Interaction Policy: Delegate, Request, Revoke and Cancel.

Basic policy. There are four types of basic policy in ReiT.

① Right Policy: The right policy specifies that the policy subject is allowed to carry out a series of actions to other agents or to the environment. It is described as follows:

Conditions Right(Subject, Action)

Among them, Conditions is an expression. The ReiT policy permits complex conditions to be built from the basic conditions and supports the following operators (and, or, not); Right(Subject, Action) represents that the Subject has the power to carry out the Action.

② Obligation Policy: The obligation policy specifies a series of actions that the policy subject must be carried out. It is described as follows:

Conditions Obligation(Subject, Action)

Among them, Obligation(Subject, Action) represents that the Subject has the obligation to carry out the Action.

③ Prohibition Policy: The prohibition policy specifies a series of actions that the policy subject is not allowed to carry out. It is described as follows:

Conditions Prohibition(Subject, Action)

Among them, Prohibition(Subject, Action) represents that the Subject is not allowed to carry out the Action.

④ Dispensation Policy: The dispensation policy specifies actions that the subject must carry out in the past, but not need to do now. It is described as follows:

Conditions Dispensation(Subject, Action)

Among them, Dispensation(Subject, Action) represents that the Subject doesn't need to carry out the Action any more.

Interaction policy. The ReiT policy language can describe the interactive behaviors. There are four types of policies offered to influence the interaction between the subject and other entities.

① Delegate Policy: A delegation policy allows an entity to give a right to another entity or group of entities. It is described as follows:

Conditions Delegate(Sender, Receiver, Right(Receiver, Action))

Among them, Sender is the sender of the right and the Receiver is the receiver of the right. Delegate(Sender, Receiver, Right(Receiver, Action) represents that the Sender gives its right to Receiver.

② Request Policy: There are two kinds of request policy: a request for an action and a request for a right. It is described as follows:

(ⅰ) request an action

Conditions Request(Sender, Receiver, Obligation(Receiver, Action))

The whole rule represents that the Receiver has the obligation to perform the Action when the request is accepted and the Conditions are true.

(ⅱ) request a right

XConditions Request(Sender, Receiver, Right(Sender, Action))

The meaning of the whole rule is that the Sender is allowed to perform the Action when the request is accepted and the XConditions of Sender are true.

③ Revoke Policy: Revocation policy is the removal of right. It is described as follows:

Conditions Revoke(Sender, Receiver, Prohibition (Receiver, Action))

It represents that the Receiver is prohibited to perform the Action when the Conditions are true.

④ Cancel Policy: An entity can cancel any request sent by the entity itself. It is described as follows:

(ⅰ) Cancel an action

Conditions Cancel(Sender, Receiver, Dispensation(Receiver, Action))

It represents the Receiver is no longer to perform the Action when the Conditions are true.

(ⅱ) Cancel a right

Conditions Cancel(Sender, Receiver, Prohibition(Sender, Action))

It represents that the Sender no longer has the right to perform Action when the Conditions are true.

Conflicts occur if policies overlap in subject, target and action but the policy objects are different. We use the priorities to resolve it. Priorities can be specified between policy rules. For instance, the predication Overrides(A, B) means that A is prior to B, and the conflict between the two rules is resolved at run-time.

Ontology component of ReiT

As we mentioned above, the ReiT policy language is not only the rules to describe the non-structural facts but also the ontology given in figure 3 to describe the structural temporal facts and the vocabulary in ontology can be appeared in rules.

To evaluate the policy, we propose a mixed reasoning framework based on the rules and ontology in the following section.

B. Mixer reasoning framework based on the rules and ontology

We present a pragmatic approach using the Jess and Racer to reason the ReiT policy. The mixer reasoning framework shown in Figure 6 consists of the following steps:

Step 1 The ontology instances involved in ReiT are mapped to Jess facts after reasoning by Racer;

Step 2 The rules in ReiT are mapped to Jess rules and facts;

Step 3 The new Jess facts inferred from the Jess rules and initial facts;

Step 4 Map the new Jess facts to rules, evaluate the policy and add new rules to ReiT knowledge.

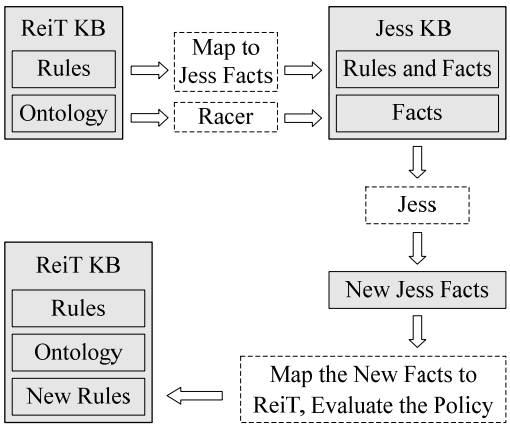

Figure 6. Mixer reasoning framework based on the rules and ontology

V. AGENT MANAGEMENT IN PAWSSF

The policy-based web service security framework mainly includes three types Agent: User Agent, Web Service Agent, and Policy Agent:

① User Agent, the function of which includes surrogating users to submit web service request call; getting users' context information and storing that in the user context database; encapsulating users' context information into SOAP message; sending the SOAP message with users' context information to Web Service Agent; receiving the result of Web Service Agent.

② Web Services Agent, the function of which includes receiving the service call requests from User Agent; parsing SOAP messages sent by User Agent and extracting users' context information; encapsulating the context information of users and web services into FIPA ACL message; sending the context information of users and web services to Policy Decision Agent; receiving the assessment result from Policy Decision Agent; calling the corresponding web service.

③ Policy Decision Agent, the function of which includes receiving the FIPA ACL message with context information from Web Service Agent; extracting the context information of user and web service; querying the web service's WSDL document from UDDI; looking up the access control policy of the web service from policy database; calling ReiT policy inference engine to assess the access policy; returning the assessment result to Web Service Agent.

In the PAWSSF, the behaviors of web services, which are represented by web services agent, are acted according to the policies. In a sense, the policy decision agent is a policy aware agent.

A. Construction of policy aware BDI Agent

We adopt the BDI agent model[11] as the basis of our agent making internal decision. Our basic approach is to use right policies to define what actions an agent is permitted or forbidden to do. The agent is responsible for ensuring that these constraints interact with its beliefs, desires and intentions at the reasoning stage, before acting towards its goals.

We propose a policy aware BDI agent whose state is viewed as consisting of mental components such as beliefs, desires, intentions, capabilities (modeled as right) and commitments (modeled as obligation). With the above notions, we propose a policy aware BDI agent.

The construction of the policy aware BDI agent consists of the five steps, as shown in the Fig.7.

Step 1 We use the rules in ReiT policy file with XML+OWL to describe the agent behaviors;

Step 2 We parse the policy file including the syntax analysis, XML parser and interpreter. The agent is responsible for obeying these policies;

Step 3 After evaluating the applicability of the policies, the ReiT rules are mapped to Jess rules;

Step 4 The internal BDI rules are translated to Jess rules; After merging the ReiT rules and BDI rules we send these rules to ReiT reasoner;

Step 5 With the reasoning result, the agent can act according to its Beliefs and Desires, constrained by the Obligations and Rights.

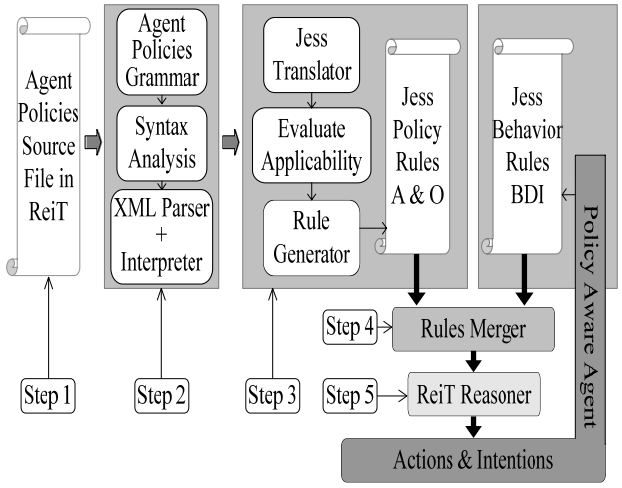

Figure 7. The construction of the policy aware BDI agent

VI. The Implementation Of PAWSSF Prototype System

A. The Implementation of prototype system classes

The classes of prototype system include: context management class, policy management class, web services class and Agent classes, such as user agent, web service agent, policy decision agent.

① context management class, which can save, get, update and delete the context information of web services, and encapsulate context information into SOAP message and FIPA ACL messages, etc.

② policy management class, which can save, get, update and delete policies, and asses ReiT policy, etc.

③ web services class, which can query the description documents of web services from UDDI, receive requests through SOAP message, and return request results, etc.

④ User agent class, which can surrogate users to submit web services requests call, get user context information and store that in the user context database, encapsulate user context information into SOAP message, send the SOAP message with user context information to web service agent, and receive the results of web service agent, etc.

⑤ Web services agent class, which can receive the service call requests from user agent, parse SOAP messages sent by user agent and extract user context information, encapsulate the context information of users and web services into FIPA ACL message, send the context information of users and web services to policy decision agent, receive the assessment result from policy decision agent, and call the corresponding web service, etc.

⑥ Policy decision agent class, which can receive the FIPA ACL message with context information from web services agent, extract the context information of user and web service, query the web services WSDL document from UDDI, look up the access control policy of the web services from policy database, call ReiT policy inference engine to assess the access policy, and returning the assessment result to web services agent, etc.

The classes diagram of prototype system is shown in Fig.8.

The system is implemented on Java EE platform[12] and JADE Agent platform[13].

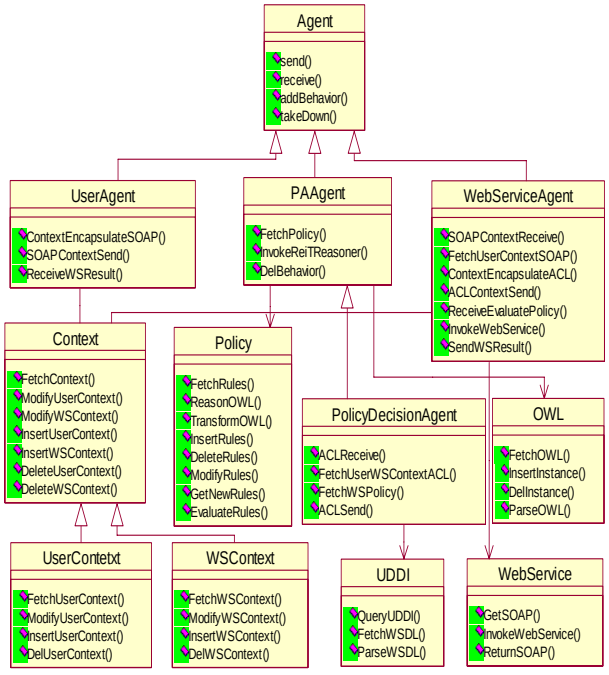

Figure 8. Classes Diagram of prototype system

B. Prototype System Performance Analysis

For web services with different sized tasks, consider the web services execution time with and without access control respectively, the run time of web services working with multi-jobs is shown in Fig.9.

For the same web services, consider the relationship between the number of concurrent users and the response time of system, the web service execution time with and without access control are shown in Fig.10.

The experimental results show that the web services security framework is reasonable and feasible.

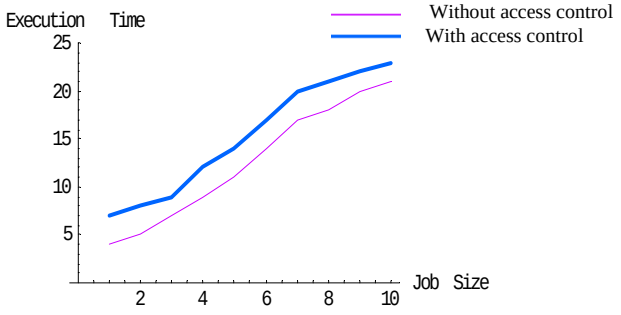

Figure 9. System response time analysis diagram

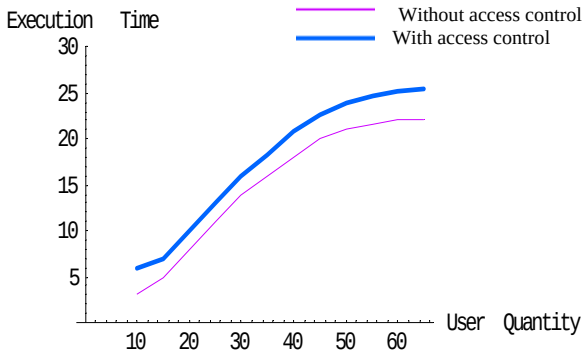

Figure 10. System Reliability Analysis Diagram

VII. A Case Study

In this section, we use a case study to introduce how to use ReiT policy language to describe the access control of the web service.

① Access control policy of web service

For a given weather forecast web service webservice1, the access control policy exists in the policy repository as follows:

(ⅰ) User must login from the given IP (202.195.48.53) to access the webservice1;

(ⅱ) User can access the webservice1 during the interval (01:00:00 - 23:00:00);

(ⅲ) User can access the webservice1 when the load of the webservice1 is less than the max load (10).

The policy of webservice1 mentioned above can be described as follows:

(LoginIP(user, uip) ∧ SameIP(aip, uip) ∧ before(t, t1) ∧ after(t, t2) ∧ WS(ws) ∧ Less(curload, maxload)) Right(user, ws)

Among them, LoginIP(user, uip) represents the user logins from the IP of uip; SameIP(aip, uip) means that address aip and uip are the same IP, and the value of aip is given "202.195.48.53"; before(t, t1) means that the instant t is before the instant t1 , after(t, t2) represents that the instant t is after the instant t2; WS(ws) means that ws is a Web Service; Less(curload, maxload) represents that the current load of the webservice1 is less than maxload, and the value of maxload is given "10"; Right(user, ws) means that user can access the ws.

Moreover, Assuming that Jimmy has the right to delegate, he delegates to Bob the right to access the webservice1 as long as he is working on the same project as Jimmy.

(SameProject(sender, receiver) ∧ WS(ws)) Delegate(sender, receiver, Right(receiver, ws))

Among them, SameProject(sender, receiver) means that sender and receiver are working on the same project, Delegate(sender, receiver, Right(receiver, ws)) represents that sender delegates the right of accessing the ws to receiver.

②User submits web service request in PAWSSF platform

Jimmy logins PAWSSF platform, the address of submitting weather forecast is: http://www.webservicex.net/WeatherForecast.asmx,

the address of providing SOAP Action is : http://www.webservicex.net/GetWeatherByZipCode,

the postcode required when query weather is stored in ZipCode.xml. As shown in Fig.11.

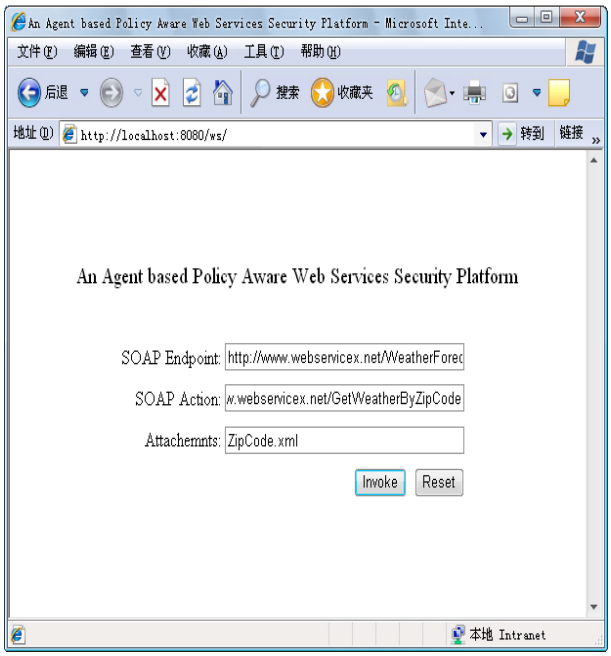

Figure11. User submit the Web Services with PAWSSF

③Agents Construction

After user agent submits request, system creates three Agents, User Agent (useragent), Web Services Agent (wsagent), and Policy Decision Agent (pdagent).

Useragent and wsagent query the context information of user and web services respectively, then pdagent assesses the access control policy of the web services. As shown in Fig.12.

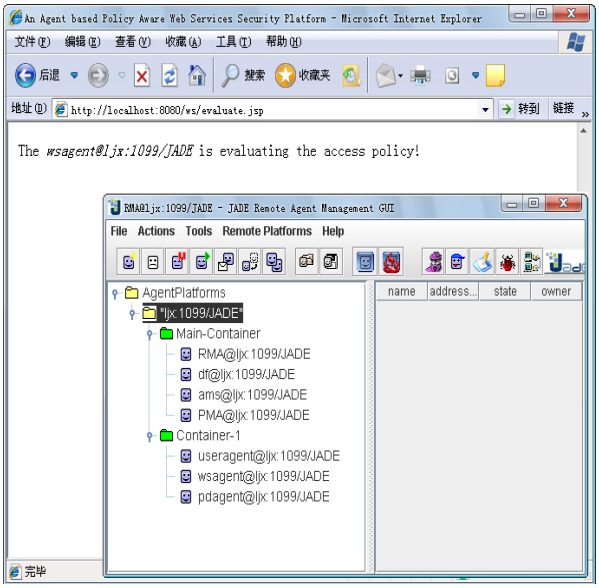

Figure12. Policy Decision Agent construction

④ Context information

Suppose that Jimmy access the webservices1 at time "22:30:00" from the IP of "202.195.48.53". And the webservice1's current load is "7".

⑤ Policy evaluation

With the context of Jimmy, it makes a call to the ReiT policy reasoning engine.

(ⅰ) Ontology reasoning

There are following facts in the temporal ontology: Instant(01:00:00), Instant(23:00:00), Instant(22:30:00), before(01:00:00, 22:30:00) and before(22:30:00, 23:00:00). Racer can infer the new fact "after(23:00:00, 22:30:00)" based on the relation between before and after in the temporal ontology.

(ⅱ) Rules reasoning

After mapping the fact "after(23:00:00, 22:30:00)" to the format that Jess supports, the new fact inferred by Jess is as follows:

Right(user Jimmy) (accws webservice1)

It means that Jimmy has the right to access the webservice1. Additional, the known fact that Bob and Jimmy are working on the same project can be represented as follows:

(assert (SameProject(sender Jimmy) (receiver Bob)))

The new fact can be inferred by Jess is as follows:

Right(user Bob) (accws webservice1)

It means that Bob has the right to access the webservice1. The value of the attribute "Effect" is modified into "Permit" after mapping the fact to ReiT.

⑥ Invoke the web service

As the result of policy evaluated by policy decision agent is "Permit", Web service agent invokes the webservice1 and returns the result to the user agent after the webservices1 invoked successfully.

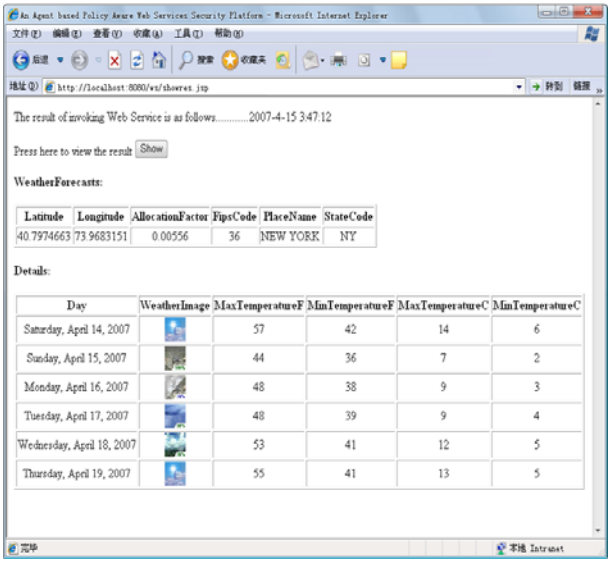

Figure13. the results returned by Web Services Agent

VIII. Related Works

Currently, there are many access control models, such as RBAC, TBAC and PBAC, etc.

RBAC model simplifies the authorization by relating the user's permission to the role of the user. However, it has the drawback of fixed role diffusion that causes increased burden on system management. From the viewpoint of task oriented, TBAC model is constructed for the application rather than the system. PBAC presents the concept to provide action before being authorized. But the behavior verification is difficult in open environment.

There are a lot of languages used to define policy, such as Poder[14], KAoS[15] and Rei, etc.

Ponder uses the object oriented idea to define policy. But the lower abstract levels provided by calling methods limits its ability. KAoS defines policies based on ontology. But it is difficult to define specific policies by using OWL alone. Although the Rei policy is described by OWL-Lite, there are no new conclusions to be drawn.

IX. Conclusion and Future Work

This paper presents a policy-based web services security framework. Innovations in this paper are as follows: ① The PAWSSF has the feature of loosely-coupled as separating the policy decision point from the policy execution point; ② The policy evaluation is based on the context of the user and web service. It is helpful to improve the policy adaptation in the open environment; ③ The ReiT policy based on rules and ontology can express the structural and non-structural knowledge; ④ We propose an effective mixed reasoning framework based on rules and ontology. ⑤ We propose a policy aware BDI agent to authorize the access control of the web services. In future work, we plan to study how to

verify the consistency between policies. Moreover, the distributed detection of the policy conflict is to be studied.

ACKNOWLEDGMENT

This paper is supported by the National Science Foundation of China under Grant No. 60903130, and the Natural Science Foundation of the Jiangsu Province of China under Grant No. BK2007074, BK2009698, BK2009699.

REFERENCES

[1] M.P. Papazoglou, W.J. Heuvel. Service oriented architectures: approaches, technologies and research issues. International Journal on Very Large Data Bases, 2007, 16(3):389-415.
[2] J. Yu, Y.B. Han. Service oriented computing-principle and application. Beijing:Tsinghua university press, 2006.
[3] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman, “Role-based access control models”, Computer, IEEE Press, Feb. 1996, pp. 29(2): 38-47.
[4] S. Oh, S. Park, “Task-role-based access control model”, Information Systems, Elsevier Science Ltd., Sep. 2003, pp. 28(6): 533-562.
[5] M. Kudo, “PBAC: Provision-based access control model”, International Journal of Information Security, Springer Berlin, Feb. 2004, pp. 1(2): 116-130.
[6] J.M. Serrano, J. Serrat; A.Galis, “Ontology-Based Context Information Modelling for Managing Pervasive Applications”, 2006 International Conference on Autonomic and Autonomous Systems(ICAS'06), IEEE Press, USA, Jul. 2006, pp. 47-52
[7] R. Nabhen, E. Jamhour, and C. Maziero, “A Policy Based Framework for Access Control”, Information and Communications Security, Springer Berlin, Sep. 2003, pp. 2003(2836): 47-59.
[8] H. Wache, T. Vogele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hubner, “Ontology-Based Integration of Information-A Survey of Existing Approaches”, 17th International Joint Conference on Artificial Intelligence(IJCAI01), USA, Aug. 2001, pp. 108-117
[9] D. Weyns, G. Vizzari, and D. Keil, et al., Environments for Multi-Agent Systems II, Springer Berlin, The Netherlands, Feb. 2006
[10] L. Kagal, T. Finin, and J. Anupam, “A Policy Language for A Pervasive Computing Environment”, IEEE 4th International Workshop on Policies for Distributed Systems and Networks, IEEE Press, Jun. 2003, pp. 63-74.
[11] J. Buford, G.Jakobson, and L. Lewis, “Extending BDI Multi-Agent Systems with Situation Management”, Information Fusion, 2006. ICIF’06. 9th International Conference, Jul. 2006, pp. 1-7.
[12] K. Mukhar, et al., Beginning Java EE 5: From Novice to Professional, friends of ED, 2005.
[13] Java Agent DEvelopment Framework Homepage, http://jade.cselt.it/, 2009.
[14] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, “The Ponder Policy Specification Language”, Workshop on Policies for Distributed Systems and Networks, UK, Jan. 2001, Springer-Verlag LNCS, pp. 18-39.
[15] G. Tonti, J.M. Bradshaw, R.Jeffers, R. Montanari, N. Suri, and A. Uszok, “Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder”, The SemanticWeb - ISWC 2003, Springer Berlin, Sep. 2003, pp. 419-437.

Bin Li was born in Yangzhou, Jiangsu Province, China, in 1965. He received the Ph.D. degree in computer application technology from Nanjing University of Aeronautics & Astronautics, Jiangsu, China in 2001.

Currently, he is a Professor of Yangzhou University and conducts research in the areas of service oriented computing, multi-agent system and artificial intelligence.

Lingjun Zhao was born in 1987, M. S. candidate. Her main research interests include service computing oriented computing, software agent.

Junwu Zhu was born in Yangzhou, Jiangsu Province, China, in 1972. He received the Ph.D. degree in computer application technology from Nanjing University of Aeronautics & Astronautics, Jiangsu, China in 2008.

Currently, he is a Associate Professor of Yangzhou University and conducts research in the areas of service oriented computing, Ontology and artificial intelligence.

Jun Wu was born in Yangzhou, Jiangsu Province, China, in 1970. He received the Ph.D. degree in computer application technology from Southeast University, Jiangsu, China in 2005.

Currently, he is a Associate Professor of Yangzhou University and conducts research in the areas of service oriented computing, computer network and formal method.