Semantic Web Service Architecture Using Multi-agent Scenario Description

Sachiyo Arai, Yohei Murakami, Yuki Sugimoto, and Toru Ishida

Department of Social Informatics, Kyoto University,
606-8501 Kyoto, Japan
{sachiyo,yohei,sugimoto,ishida}@kuis.kyoto-u.ac.jp

Abstract. Current research issues on web services have come to center around flexible composition of existing services. Under the initiative of industry, flexible composition framework has been developed on a workflow model where flow of the processes and bindings among services should be known beforehand. In short, its framework realizes flexible composition within the available services that are not widely opened. This paper focuses on two limitations of current web service composition. One limitation is that it's hard to represent multi-agent scenarios consisting of several concurrent processes because it is based on a workflow model. The other limitation is that once composed, web service cannot be reused or transferrd for other requesters, because there is no function to put the semantics on composite web service. To overcome these limitations, we have developed scenario description language *Q*, which enables us to realize a web service composition reflecting a multi-agent's context not as a workflow but a scenario. Furthermore, we developed a system that translates multi-agent scenario to DAML-S, and that registers the translated DAML-S as a new Web service. We also discuss the availability of our system for designing an application to C-Commerce and Digital Cities.

1 Introduction

Web technologies, such as Universal Description, Discovery, and integration (UDDI) [UDDI 2002], Web Services Description Language (WSDL) [Christensen 01] and Simple Object Access Protocol (SOAP) [Box 00] have extended people's usage of information on the Web. The most recent technologies, such as Business Process Execution Language for Web Services (BPEL4WS)[Virdell 03] and Web Service Choreography Interface (WSCI)[Arkin et.al 01] realize a dynamic composition of web services. However, there remain some difficulties within these frameworks to realize a fully dynamic and flexible composition [Benetallah 02].

We focus on two difficulties of current web service composition. One is that a workflow model is not good enough to represent composition of web service, where there include concurrently executable service components. In other words, describing compositions of web services as a workflow seems harder than describing them as a multi-agent scenario, where concurrently executable services can be easily represented. The other is that above frameworks do not have a function to insert semantics into the composite web service. The lack of this function not only restricts

J. Lee and M. Barley (Eds.): PRIMA 2003, LNAI 2891, pp. 98–109, 2003.

combinable services to several providers who have the same semantics, but also restricts the reusability of composite web services that was once made, to several requesters who understand its semantics. Although a large number of studies have been carried out on the latter problem, which is usually called an 'interoperability' problem [Martin 99][Nodine 98][Sycara 99], little attention has been given to the former one. We focus on these two problems.

In dealing with the former problem, we have developed a scenario description language *Q* and IPC (Interaction Pattern Card), which provide the environment where the web services can be easily composed in terms of a multi-agent scenario that controls agents' behaviors [Ishida 02b]. The advantage of applying this language to web service domain is twofold. One is that it enables users to easily describe web service composition, even if it includes concurrently executable services. The other is that it can handle multi-agent's scenario consisting of multiple service requesters who need cooperation as well as locally independent processing on the web.

Digital cities [Ishida 02a], which are being developed around the world, and collaborative commerce (C-commerce) [Behruens 00], which has recently become a hot issue in e-market place, are good examples of our objective domain where the scenario description works more effectively than the workflow model. Because there exist heterogeneous agents that behave concurrently as well as cooperatively in these domains, a multi-agent scenario will be woven through "interaction design" of which importance would be claimed in this paper.

As to the latter problem, we expect the Semantic Web [TBL 02] to give well-defined meaning to better enable computers and people to work in cooperation. This would be realized by marking up the contents of Web with well-defined semantics. Our work adopts DAML-S [Ankolokar et.al. 01] as service description language because it provides a semantically based view of a web services. Though this language enables agents to automatically discover, evaluate and invoke web services, it is too difficult for human users to write this language. Therefore, we introduce a translator, which translates the multi-agent web service scenario, written by our language, to the DAML-S, to put semantics on the scenario for the purpose of providing flexible reusability of once combined web services.

In this paper, we introduce a system which adopts a Semantic Web approach and scenario description language to guarantee interoperability and interaction design, respectively, and show their performance by applying them to the domain of composing web services. In the following sections, we discuss our philosophy of designing multi-agent system, which supports human user on the Web. Particularly, we stress the importance of "interaction design" among the agents. Section 3 introduces the scenario description language *Q*, and shows application domain where a multi-agent scenario would be necessary. Section 4 describes our proposed architecture for the web service composition and explains about the translator that adds semantics to the web service composition. We conclude in Section 5.

2 Design Philosophy

We consider three requirements that architecture of web service composition should fulfill. First, it should provide a framework for 'interaction design' among heterogeneous and legacy agents with different roles. Here, each agent plays a

different role but needs collaboration to achieve their respective goals. Second, it should guarantee 'interoperability' among heterogeneous agents from different platforms. Third, it should provide a framework to put semantics on the web service composition for the purpose of providing flexible reusability of once combined web services. Here, we will discuss previously developed systems by industries' initiative, and introduce our standpoint of designing web service architecture where "human is in the loop".

2.1 Current Web Services

Interaction Design in WSFL, XLANG, and BPEL4WS

Interaction design has been developed for web services, such as IBM Web Services Flow Language (WSFL), and Microsoft's XLANG. WSFL specifies interactions between web services which are modeled as links between endpoints of the web services' interfaces. Microsoft XLANG provides both the model of an orchestration of services as well as collaboration contracts between orchestrations. BPEL4WS [Virdell 03] provides a language for the formal specification of business processes and business interaction protocols. It is designed to ensure that differing business processes can understand one another in a web services environment, and that they can realize a dynamic composition, but its components are limited to a several static service providers.

In the framework mentioned above, interactions among the components of web services are described as a workflow. However, since concurrently executable service components are included within a composition, it is hard to represent their interactions as a workflow. In other words, it seems harder to describe compositions of web services as a workflow than as a multi-agent scenario described in an event driven manner where concurrently executable services can be easily represented through the 'interaction design' processes.

Interoperability in UDDI, WSDL, and SOAP

Web services are designed to provide interoperability between diverse applications. The platform and language independent interfaces of the web services allow the easy integration of heterogeneous systems. Web languages such as Universal Description, Discovery, and Integration (UDDI), Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP) define standards for service discovery, description and messaging protocols. In these frameworks, since the common interfaces are defined within the similar services, it is easy to find an alternative service to be combined. In addition, UDDI descriptions are augmented by a set of attribute, called tModels, which describe additional features such as the classification of services. However, these standards and protocols are available within several static service providers but don't contribute to an interoperability for a web service recomposition.

As mentioned above, current platform is hardly adequate neither for realizing complex composition of services on the Web, nor utilizing Web's resource to it's fullest. Notably, architecture for (1) unfolding E-commerce, which is carried out by multiple enterprises or binding dynamic collaboration processes to coordinate information and services of cities in an open environment, (2) registering composite

process, which is generated as a result of Web service composition, to the Semantic Web is needed.

2.2 Interoperability versus Interaction Design

The two approaches of interoperability and interaction design are complementary for multi-agent design (see Fig. 1).

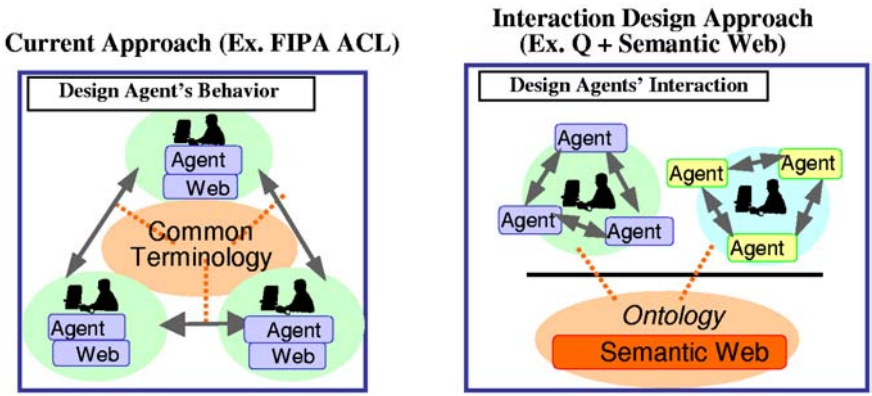

Fig. 1. Collaboration on the Web through Interaction and Interoperation

Interoperability

In order to guarantee interoperability among agents from different platforms, several specifications have been proposed by FIPA. Agents must be designed to be compliant with these specifications in order to interoperate with each other. Among these specifications, the Agent Communication Language (ACL) was designed to allow agents to communicate with each other. Therefore, we focus our discussion on ACL for interoperability among agents. Speech Act theory has been widely accepted in ACL as a way for designing agents that can react to a message, because an agents knows its illocutionary function while it may do not understand the propositional content. Doing so, ACL can be independent from content language. Therefore, a lot of ACL is based on speech act theory, including FIPA ACL and KQML [Finin 94].

FIPA ACL contains a set of message elements, including performative, sender, receiver, reply-to, content, language, encoding, ontology, protocol, conversation-id, reply-with, in-reply-to and reply-by. These elements realize interoperability among the very limited web services which are based on FIPA ACL framework.

Interaction Design

Interaction design has been widely studied for web services IBM WSFL specifies interactions between web services which are modeled as links between endpoints of the web services' interfaces. Microsoft XLANG provides both the model of an orchestration of services as well as collaboration contracts between orchestrations. BPEL4WS is designed to ensure that differing business processes can understand one another in a web services environment. Here, the interactions among the web service

components, which are regarded as the agents, are described as a worklow where it seems hard to describe the interaction among the concurrently executable agents.

We introduce *Q* to describe a *human request* to a large number of (*legacy*) *agents* with different roles [Ishida 02b]. Its primitives include *cue* and *action*. A sensing function is defined as a *cue*, and there is no side effect to sensing. An acting function is defined as an *action*, which may change the environment of agent system. The *cues* and *actions* are performed by agents, and their semantics is dependent on those agents. A Guarded command is for describing concurrent reactive behavior of agents. A scenario is described as state transitions. Each state is defined by a guarded command. The Scenario Description language can provide a clear interface between agent system developer, scenario writer and interaction designer.

Collaboration between Scenario Writer and Agent System Developer: The procedure of creating a new scenario is described as follows. First, the scenario writer and agent system developer agree upon the *cue*s and *action*s as an interface between them. Second, the scenario writer describes a scenario, while the agent system developer implements *cue*s and *action*s. The biggest effect of introducing *scenario* is that it can provide a clear interface between computer professionals and application designers, who have very different perspectives, and this effect would contribute to realize dynamic composition of web services without middle-agent frameworks. [Sycara 99] [Martin 99] [Nodine 98].

2.3 Complementary Approaches

As discussed above, the two approaches of interoperability and interaction design are complementary for multi-agent design, which means how to compose the web services.

Collaboration among Agents and Collaboration among Humans: The interoperability approach aims at collaboration among autonomous agents from different vendors and platforms, while the interaction design approach allows humans from different perspectives to collaborate when designing agents which crawl across the web on behalf of humans.

Interoperation among Agents and Interaction between Human Users and Agents: Agent communication language (ACL) aims at allowing agents to communicate with each other, while multi-agent scenarios describe the interaction between human users and agents. Agents constrained under scenarios can interoperate with each other by ACL.

Interoperation and Architecture Design of Agents: Composing web services can be considered as a multi-agent planning. Here, it is rather difficult for heterogeneous agents to decompose problems into sub-problems, allocate sub-problems, exchange sub-problem solutions and synthesize overall solutions autonoumously, even if the interoperation does work among the agents. Agents should be intelligent and autonomous in interoperability approach to achieve their goals. Whereas with the interaction design approach, multi-agent planning might be done by human users, who request the web services, by means of scenarios that coordinate agents activities.

In interaction design, agents can be autonomous or dependent. If autonomous, scenarios can be simple; if not, scenario writers should specify all details.

In the next section, we introduce the language we have been developed to describe multi-agent scenarios to realize web services composition.

3 Languages for Multi-agent Scenarios

3.1 Interaction Design Language: *Q*

The language *Q* is originally designed for describing scenarios of multi-agent's. For details see [Ishida 02b]. We currently re-designed *Q* for describing a variety of web service composition. In the multi-agent world, it requires to control multiple agents by the multi-agent scenario in the distributed environment. In the following section, we introduce the scenario description language *Q,* and discuss how multi-agent scenario is described for web service composition. The salient features of *Q*'s language functionality are summarized as follows. To explain the feature *Q*, we take an example of a simple Web service composition scenario as shown in Fig.2. Here, the Web service takes user's U.S. address to find the ZIP code and temperature.

Cues and Actions: An event that triggers interaction is called a *cue*. Cues are used to request agents to observe their environment. No cue is permitted to have any side effect. Cues keep on waiting for the event specified until the observation is completed successfully. Comparable to cues, *actions* are used to request agents to change their environment. The line starting with !verb denotes a request to the agent to make a certain action, while the line starting with ?verb denotes request to make a certain observation. Types of cues and actions are limited to ones given a priori, but unlike functions, the content of the cues and actions need not be defined since the interpreter carries out the interpretation and scenario execution. Thus, scenario can be said to have high readability, since it is complete in itself.

Scenarios: Guarded commands are introduced for the situation wherein we need to observe multiple cues simultaneously. A guarded command combines cues and actions. After one of the cues becomes true, the corresponding action is performed. A scenario is used for describing state transitions, where each state is defined as a guarded command. Scenarios can be called from other scenarios. Scenario describes state transitions using guarded command. Through scenarios, describing interactions according to states becomes possible. By using '*go sentences'* to define the next state, state transition can be described. If executing a group of actions end without specifying the next state, then it means the state transition is completed.

Agents: Each agent is defined by a scenario that specifies what the agent is to do. Even if a crowd of agents executes the same scenario, the agents exhibit different actions as they interact with their local environment (including other heterogeneous agents).

```
(defagent sample-agent
        sample-scenario)
   (defscenario sample-scenario (&pattern ($state "")
                                ($city "")
                                ($address "")
                                ($zipcode "")
                                ($temperature "")) 
    (scene1
     (#t
      (!open-input-window :label '("address" "city" "state"))
      (go scene2)))
    (scene2
     ((?push-button :button "OK")
      (!get-text :label "address" :result $address)
      (!get-text :label "city"    :result $city)
      (!get-text :label "state"   :result $state)
      (!webservice :wsdl-file
                   "http://webservices.eraserver.net/zipcoderesolver/zipcoderesolver.asmx?WSDL"
                :operation "ShortZipCode"
                :input (list $address $city $state)
                :extract '("ShortZipCodeResult")
                :result $zipcode)
      (!webservice :wsdl-file "http://www.ejse.com/WeatherService/Service.asmx?WSDL"
                :operation "getTemp"
                :input $zipcode
                :extract '("return")
                :result $temperature)
      (!display :data (list "Address:" $address $city $state
                        "Zip Code:" $zipcode
                        "Temperature:" $temperature))
      (go scene1))
     ((?push-button :label "End")
      (!display :data (list "Bye!")))))
```

Fig. 2. Multi-Agent Web Service Scenario

3.2 Application Domain

The language Q realizes a collaboration needed in C-commerce (Collaborative commerce), a next generation E-commerce (which is referred to as next generation E-commerce), and in Digital City.

Collaborative Commerce (C-Commerce) is the name given to commercial relationships carried out over a collaborative framework to integrate enterprises' business processes, share customer relationships and manage knowledge across enterprise boundaries [Behruens 00]. The ultimate aims of C-Commerce initiatives are to maximize return on intellectual capital investment, business agility and the quality of the customer experience. C-Commerce is far more crucial than basic B2B e-commerce that is designed to construct a virtual link for a pre-defined community of trading partners to buy or sell goods and services. While, the traditional web service frameworks cannot help C-Commerce focusing a one-dimensional transaction, there is no opportunity to negotiate enhancements to products in order to have them match unique customer needs and requirements. Lacking an open, reliable platform, traditional c-commerce vendors could not develop flexible, sharable drag-and-drop

modules among their products. Our proposed Semantic Web service architecture, which consists of two frameworks, such as "interaction design" and "interoperability", enables C-Commerce to offer new products, services and multi-dimensional collaboration which brings global enterprises a step closer to realizing the promise of increased velocity in supply chains and efficient inter-enterprise processes.

Digital Cities is to build an arena in which people in regional communities can interact and share knowledge, experience, and mutual interests [Ishida 02a]. Digital cities integrate urban information (both achievable and real time) and create public spaces in the Internet for people living/visiting the cities. Digital cities are being developed all over the world because the Web enables us to create rich information spaces for everyday life as well as the global businesses. While the Internet provides global information, life is inherently local and heterogeneous reflecting the different cultural background. Here, we don't need or have any standard of protocol for communication on the Web, but need some collaborative information sharing among different cities. Since this type of collaboration seems to be realized by local interaction among some cities which have a common background, rather than by global interoperation through the Web. For example, a project to link Digital City of Japan and China has commenced as part of Asia Broadband Plan. Coordination of the information retrieval agent, which is independently developed on Digital Cities, is a good example to show that application area of Web service composition is expanding.

Similar to C-Commerce, collaboration between Digital Cities requires a framework that can create cooperating task scenario, and execute that scenario using information provided on the Web.

4 Architecture for Multi-agent Semantic Web Services

In this section, we will explain how to convert scenario description to DAML-S, which put the semantics to existing web service composition scenario, enabling other users to flexibly recompose or reuse those services. In the following, we introduce our Semantic Web service architecture which consists of two main modules; one is for realizing Multi-agent scenario, and the other is for translating it to DAML-S description to add the semantics.

4.1 Architecture

By means of converting a web service scenario, which based on *Q,* to the representation of description logic, semantics of service composite process is described.

The description of composite process added semantics is called the 'process semantics'. The 'process semantics' enables other service requesters to utilize this service composition for a web service plan to be satisfy this requester's constraints, using concept layers provided by service ontology. We employed here DAML-S, as an ontology language that describes property and capability of web services. Figure 3 shows our web service architecture for creating, converting, and releasing web service scenario on the Web.

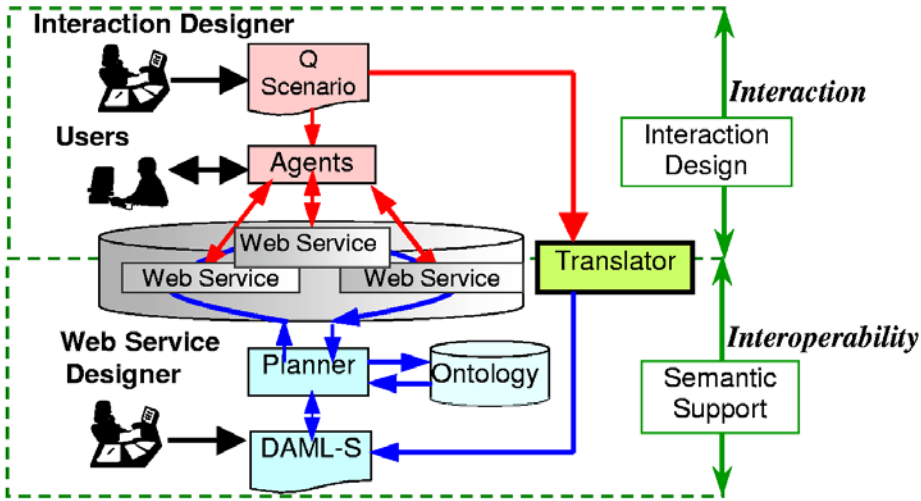

Fig. 3. Semantic Web Service Architecture for Composing Multi-agent Scenario

4.2 Scenario Execution by Multiple Agents

The executor for Web service scenario constitutes of Web service interface written in *Q* language interpreter and Scheme. Figure 4 shows the system architecture. Components surrounded by bold lines indicate Web service interface. Cues and actions described in the scenario is interpreted and executed by the interpreter. In case an action executing a Web service is given, Web service interface is called.

Information needed to execute Web service, such as URL in WSDL file, operation that is used, and value(s) to be sent to the service, are included in the interpreter's execution request. The interface first executes parser for analyzing WSDL. The parser obtains WSDL document from assigned URL, and then analyzes the document. When it finishes analyzing the document, outgoing SOAP message and address of SOAP server is extracted. Following this, the interface exchanges messages with Web service's SOAP server at SOAP transmitter/receptor. When the result is returned, the data is returned and then the procedure is terminated.

4.3 Scenario Translator to DAML-S Description

By describing service composition process in *Q* language, we have enabled users to easily construct a composite service. However, processing transaction between multiple, different Web services cannot be realized only with Web service composition scenarios. Moreover, since service scenarios lack reusability services accommodating situations of various users cannot be provided. By converting scenario descriptions, which reflect the needs of users in various situations, to DAML-S, reusing the scenario becomes possible through placing Web service composition mechanism on the server-side. In this section, the conversion of 'guarded command' to DAML-S process semantics is outlined. Guarded command is important

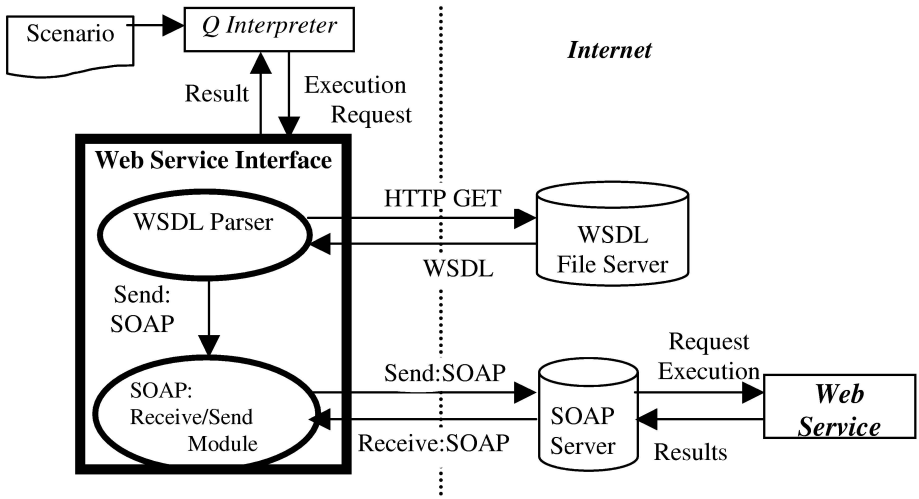

Fig. 4. Scenario Execution

in realizing multi-agent scenario, and converting multi-agent service scenario into DAML-S process semantics must be accurately performed. But we don't yet to examine the completeness of this translation.

Guarded command generally is expressed in multiple pairs of guard and statement. A guard consists of Boolean function and communication primitives such as transmission or reception. When Boolean function is true and communication primitive is ready to communicate, the guard is considered true. If guard is considered true inside the guarded command, then next execution line is executed. However, if multiple guards are true, only one guard is non-deterministically selected to execute the next execution line since multiple guards are evaluated in parallel.

In *Q* language, event-waiting '*cue*' corresponds to a guard, whereas outside '*action*' corresponds to execution line. As mentioned in 3.1, the event-waiting corresponding to guard is considered true when event is observed, and as a result, subsequent group of actions are executed.

Guarded Command

Rules to convert guarded commands, which consist of actions and event-waiting cues, to DAML-S process model will be discussed. Guarded commands are sequentially controlled and executed in the following steps.

1. Initiate parallel execution of multiple event-waiting classes (*cue*) using split class execution. (Event observation and execution processing unit)
2. Wait until at least one event among the observed events occurs using Repeat-Until class. (Event-waiting processing unit)
3. Select one event that has occurred, and execute Action Set class that is triggered by that event, using Choice class. (Event selection and execution processing unit)

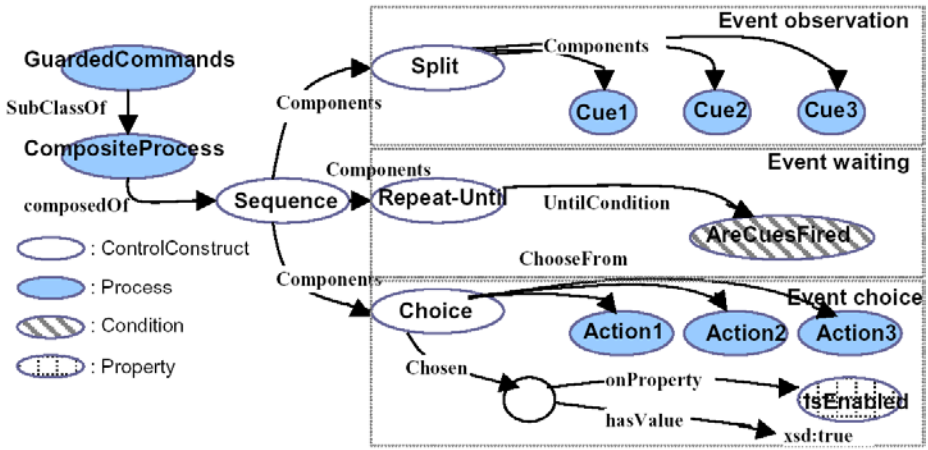

Fig. 5. RDF directed graph of Guarded command based on DAML-S Control Structure

Using control structure class called Sequence class provided in DAML-S, guarded commands are converted to subclass of Composite Process class, which sequentially executes event observation and execution processing unit, event-waiting processing unit, and event selection and execution processing unit. Figure 5 shows DAML-S expression of guarded commands, which consist of event observation and execution processing unit, event-waiting processing unit, and event selection and execution processing unit, in RDF's directed graph.

5 Conclusion

Our research aims at realizing social knowledge through sharing and employing information on the Semantic Web. As a first step, we proposed a scenario description language *Q,* which can describe multi-agents' Web service composition process through multi-agent scenarios. This language provides the framework for 'interaction design', which enables us to design interaction between users, between users and agents, and between agents, as well. Furthermore, we provided a framework that enables other users to flexibly reuse composite services by developing a translator that converts multi-agent scenario into DAML-S and a feature that registers services to the Semantic Web as a new service. By employing this translator, 'interoperability' is guaranteed on the composite web service, and consequently it becomes reusable to other users. The concepts of 'interaction design' and ' interoperability' work complementarily for coordinating heterogeneous agents and humans who have different motivations. This overall architecture not only enhances collaboration between users on the Web, but also facilitates user-agent interaction design, and therefore is expected to contribute to Collaborative Commerce and Digital City that will unfold in the Internet in the future.

References

[Ankolokar et.al. 01] A. Ankolokar et.al., DAML-S: Semantic markup for web services. *In Int. Semantic Web Working Symposium*, pages 411–430, 2001

[Arkin et.al. 02] Assaf Arkin et. Al, Web Service Choreography Interface (WSCI) 1.0, http://www.w3.org/TR/wsci/, 2002.

[Behruens 00] Stefan Behruens, Electronic Collaborative Commerce – An Overview of Enabling Technologies – Seminar Paper, Schloss Reichartshausen am Rhein, European Business School, 2000.

[Benetallah 02] B. Benatallah, M. Dumas, M.-C. Fauvet, and F. Rabhi. Towards Patterns of Web Services Composition. In S. Gorlatch and F. Rabhi, editors, Patterns and Skeletons for Parallel and Distributed Computing. Springer Verlag, UK, Nov 2002.

[Box 00] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte, and D.Winer. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/, 2000.

[Christensen 01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Desctiption Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, 2001.

[Finin 94] Finin, T., Fritzson, R., McKay, D., McEntire, R.: KQML as an Agent Communication Lnguage. *International Conference on Information and Knowledge Management (CIKM-94)*, 1994

[Ishida 02a] Ishida, T.: Digital City Kyoto: Social Information Infrastructure for Everyday Life. *Communications of the ACM (CACM)*, Vol. 45, No. 7, pp. 76–81, 2002

[Ishida 02b] Ishida, T.: *Q:* A Scenario Description Language for Interactive Agents. *IEEE Computer*, Vol.35, No. 11, pp. 54–59, 2002

[Martin 99] D. Martin, A.Cheyer, and D. Moran, The Open Agent Architecture; A framework for building distributed software systems. Applied Artificial Intelligence, 13(1-2): pages 92–128, 1999.

[Nodine 98] M. Nodine and A. Unruh, Facilitating Open Communication in Agent Systems. In M. Singh, A. Rao, and M. Wooldridge, editors, Intelligent Agent IV: Agent Theories, Architectures, and Languages, pages 281–296, Springer-Verlag, 1998.

[Sycara 99] K. Sycara, M.Klusch, S. Widoff, and J. Lu, Dynamic service matchmaking among agents in open information environments, *SIGMOD Record (ACM Special Interests Group on Management of Data),* 28(1): pages 47–53, 1999.

[TBL 02] T. Berners-Lee, J. Hendler, and O. Lassila, The Semantic Web. Scientific American, 284(5): pages 34–43, 2001.

[UDDI 02] The Evolution of UDDI Technical White Paper, http://www.uddi.org/pubs/, 2002.

[Virdell 03] Margie Virdell, Business Processes and Workflow in the Web services world, - A workflow is only as good as the business process underneath it www-106.ibm.com/developerworks/webservices/library/ws-work.html