Automating Design Support in Supply Chains on Semantic Web Services

Incheon Paik
School of Computer Science and Engineering
University of Aizu
Aizu-Wakamatsu, Fukushima, Japan
+81-242-37-2796
E-mail: paikic@u-aizu.ac.jp

Hiroyuki Akimoto
School of Computer Science and Engineering
University of Aizu
Aizu-Wakamatsu, Fukushima, Japan
E-mail: akimoto@ebiz.u-aizu.ac.jp

Shinjirou Takami
School of Computer Science and Engineering
University of Aizu
Aizu-Wakamatsu, Fukushima, Japan
E-mail: takami@ebiz.u-aizu.ac.jp

Abstract

The integration of four design attributes (component-cost, quality, function, and technology) streamlines product design by providing a multilateral view to designers that leads to cost savings in the supply chain. Since the existing information infrastructure, based on server-side components for these design attributes, does not provide interoperable Web interfaces or semantic service descriptions for agents, improved architecture for the semantic Web service environment where automating agents can work more efficiently is suggested. Based on an improved ontological paradigm which integrates product design attributes organically to improve agent efficiency, the building of a Web service environment for agents in DAML-S, which wraps the existing server-side software components to access attribute instance data, is explained. A prototype of an application-centric agent for design support in this environment and its evaluation are also described to illustrate the effectiveness of this Web service environment.

1. Introduction

As semantic Web services evolve toward complete environments, they enable the Web to provide machine-readable contents and paradigms. The Supply Chain Management (SCM) of e-businesses, with its distributed architectures [1] in such an environment will benefit greatly, as complex processes and transactions become more automated.

Viewing SCM broadly, careful consideration of the planning and design stages of product development is important to avoid unnecessary costs incurred by redesign and modification. The cost of development increases rapidly at each stage from the initial stages of development to the final stages of actual mass production. Expenses for design and planning in the initial fluid stages may not be very high but rise rapidly for the construction of production lines and for final mass production [2].

To increase the efficiency of product design for the SCM process, existing methodologies have considered separately or partially the four product design attributes of component-cost, quality, function, and technology. However, integrating these four allows multilateral inspection of all product design attributes enables greater efficiency in the design process. In order to accomplish this, an information infrastructure with new table structures that combine the four attributes and their relationships organically was recently devised [2]. The architecture has server-side components for the fundamental infrastructure of integrated design attributes, making it possible to compose application software easily and create interfaces for online application developers with some special access permission. However, these integrated design attributes are most effective when they are open to general users and can be processed by automating agents. Existing architecture cannot provide interoperable Web service interfaces that have consistent semantic metadata for general users and agents. Existing architecture can provide simple ontological concepts that can cover the definition of domain concepts for product attributes, but it still cannot provide service descriptions for agent activities or general semantic derivation which are suggested for semantic Web services [3][4]. The goal of the research described here was to develop a semantic Web service environment that can provide interoperable service interfaces for integrated design attributes to agents and general users. The entire architecture consists of several layers (Figure 2): **infrastructure** (this layer includes an ontology and instances or a relational database), **business logic**, **Web service**, and **application**. This paper focuses on the Web service environment (**Web service**), which is based on existing information infrastructure and uses the ontology for design attributes to enable efficient agent activities (**infrastructure**). A prototype of the agent in this environment and its evaluation is discussed as well.

2. Scenarios and Motivation

Presently, when companies make a product, they normally consider the quantity, price, and delivery of components for products in the supply chain. When companies design an innovative product, they consider combinations of the four product design attributes. Integrating these four attributes into one infrastructure yields positive effects on design in the supply chain [2]. To begin our explanation, consider the two scenarios below.

Scenario A. A flashlight company has the plan to increase the quality "Brightness" of its existing flashlight from 7.5cd/mm^2 to 10cd/mm^2. The company wants to know which components must be changed, what technologies will be required, and what costs will be incurred. To discover the answers to these questions, a designer must look for several product attribute databases distributed along the supply chain network with cross references (e.g., quality-component or quality-technology).

Scenario B. A flashlight company designs an innovative flashlight product with the following new specifications. Quality of brightness is greater than 7.5 cd/mm^{2,} and the flashlight has a long lifetime (i.e., the filament strength achieves a rating of more than 95% on an endurance of impact test, air tightness is greater than 0.2 in air pressure studies, and the strength of the plastic proves to be excellent). Other specifications include a medium price (i.e., the total component-cost is between 99 and 200) and suitability for young people (i.e., the level of brightness can be adjusted and the transparency of the plastic must be lower than 88). Surely, in order to find component-cost combinations and technologies needed to satisfy these requirements, the four product attributes of component-cost, quality, function, and technology as well as their cross references must be known at the same time. If it is possible to find answers to fulfill the requirements on the worldwide supply chain network by agents automatically, this would make design very efficient – a genuine need for industries. An information infrastructure which integrated these four attributes, developed in previous work, aimed at addressing the first stage of this need. Several problems arose, though, in relation to agent work on this infrastructure due to the nature of the current Web environment. These problems included interoperability, semantic metadata, consistency, and openness; however, these are now being addressed more successfully in semantic Web services [3][4].

To accomplish what was described in the scenarios using Web services, we may consider two approaches: one is to use a Web service composition that consists of elementary Web service interfaces for separate design attribute databases [5] (i.e., the case of not having integrated design attributes) and the other is to provide elementary interfaces with composite functions in the infrastructure (i.e., the case of having integrated design attributes). The first approach would reduce development costs, but would suffer from the complex Web service composition work and poor performance, and the second approach would yield better performance and scalability in exchange for initial high development costs. The research reported in this paper employs the second approach. We enlarge the concept of integrated product attributes in previous work with an additional ontology to decrease the work of Web service composition at higher layers.

The existing information infrastructure provides server-side component interfaces which use Remote Method Invocation / Internet-Inter ORB Protocol (RMI/IIOP), so business application developers can use the component interfaces for internal applications through this protocol and Web application developers can invoke these interfaces through Web interfaces like Servlet. However, some developers who require access to this design attribute data from any location cannot use these interfaces because of communication limitations due to security measures. Furthermore, some agents have no knowledge how to use these interfaces. In this research, we build a semantic Web service environment that provides open Web service interfaces and service descriptions which support automating agents and composite services.

3. Ontology of Integrated Design Attributes

An ontology provides taxonomies and captures semantic features for a domain. The ontology of integrated design attributes contributes to the construction of a fundamental semantic metadata framework for the information infrastructure for product design attributes.

To construct a table of the necessary design attributes, we need first to build four basic tables, and then to integrate all of them through a relation table. Relation tables describe the associations between these four tables, to allow organic integration and additional ontology to streamline agent activities. Each design attribute table can be converted into a relational database table or real instances of design attribute ontology.

3.1 Design attributes tables

The four basic tables would be the following: component-cost, quality, function, and technology. The component- cost table describes the composition of a complete product, which can be hierarchically broken down from the root node (complete product) into its subcomponents and final material sources through the supply chain. The final cost of a product can thus be calculated from the costs of every subcomponent. The function table consists of functions broken down into smaller tasks via functional analysis using value engineering, which defines functions related to the product and creates a functional system diagram and component-cost table classified according to functions. The quality table consists of qualities that have been broken down into smaller quality units using a Quality Functional Deployment (QFD) technique, which maps user demands on the product onto alternative characteristics, defines the design quality of the final product, and develops relations systematically between items (functional part, quality of the components, and so on) related to quality. The technology table lists the techniques or means, which can be identified by simple terms in the database, that represent the addition of human work to natural things in order to make them more useful. Here, we store technical names and technical contents in a database describing the technologies used previously for existing products. If technologies are intended for a new product, technical deployment similar to QFD is necessary. Detailed construction methods and examples are described elsewhere [2]. We provide a Web demonstration of the information infrastructure of these four design attributes at http://emplace.u-aizu.ac.jp/DSASCM/.

For the relation tables, all combinations of the four basic tables need to be considered. We defined relation levels between the items in every table. For example, the relations between component and function are described as the importance of the component in a function hierarchy with five levels, i.e., the cost of a component to support some function can be described in the component-cost-function table. The other tables are component-quality, component-technology, function-quality, quality-technology, and function-technology. These relation tables connect the four basic tables in a way that enables cross referencing among the design attributes. The ontology on these four basic attribute tables and six relation tables, with semantics and properties for the product design attributes, provides the information infrastructure with the semantic taxonomies that is needed to support product design in SCM.

3.2 Ontology design perspectives

Product attribute ontology design can be viewed from higher perspectives, such as semantic meanings or logical reasoning; however, in this paper, we focus on two pragmatic perspectives: as a definition of taxonomies in the domain to integrate the four design attributes with relations, and as an another approach to additional ontology design for improving the activities of agents.

To illustrate the first perspective, we introduce a short scenario of ontology construction for product design attributes after refinement procedures: *When a company designs a new product or upgrades an existing product, they use four basic attributes – component- cost, function, quality, and technology – and the relations between these attributes for the development. To do this, companies use the 10 information schemes described above. Some information can exist in their local DB, and the rest can be distributed throughout the supply chain network. Although the ontology can cover the distributed supply chain network, every local homogeneous information infrastructure is provided with complete database information for the 10 tables through an expansion process that calls an external interface with uniform resource identifiers (URI) to provide information services for outside product attributes.*

To illustrate the second perspective, when we design an ontology for domains or applications, if we combine the associated objects with their relations in the ontology, agents will use this ontology more effectively later. For example, if bus, train, and airplane are necessary when traveling from rural Aizu-Wakamatsu, via Tokyo, to San Francisco, we can prepare a relational ontology for combining the schedules of these three transport types. This will be efficient for the agents who work on this ontology, and it will yield the same effect as that obtained through converting a composite process, composed of several atomic processes, into one atomic process with the same function in a DARPA Agent Markup Language-Service (DAML-S) description. In our ontology, inter-attribute relations such as RelationScheme, RelationAttribute and their subclasses give agents a connection infrastructure for efficient retrieval of information.

Our ontology for product attributes consists of four kinds of top-level groups (Figure 1). The introductory group describes existing product classes and abstract classes for design attributes. The basic attributes group contains classes for the four design attributes and their sub-attributes. The relation attributes group describes the six inter-attribute relation tables for each basic attribute. The scheme group provides skeletons of instances for attribute information. Finally, the instance data from this ontology can be created by users or generation tools that extract design attributes from the existing SCM solution.

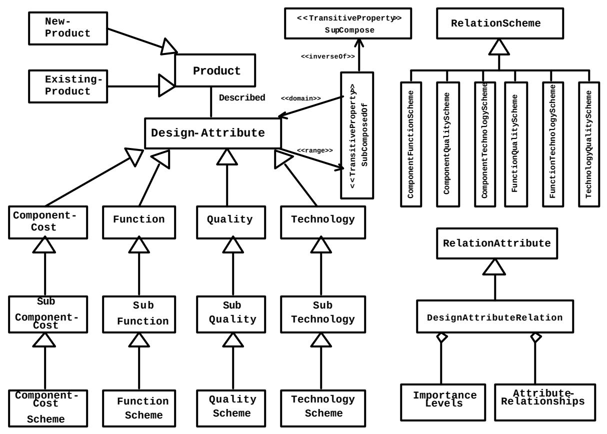


Figure 1. An ontology for product design attributes.

4. Web Service Construction

4.1 Business logic with server-side component

Online business logic requires a scalable and high performance infrastructures to manage business transactions. Since server-side middleware components such as J2EE's Enterprise JavaBeans (EJB) supports not only strong business transactions with security and directory services but also flexible software architecture for fast development, EJB was used as the basic architecture for the information infrastructure to enable access to the design attributes. Better architecture for software components can be developed via Component-Based Software Development (CBSD). One of the main advantages of CBSD is that more efficient interfaces of components can be designed, which can also become the interfaces for Web services. These interfaces must be defined carefully to ensure better system architecture and performance.

Component Interfaces. After CBSD process, five core components (AttributeTable, RelationTable, InformationGate, AttributeInstanceTable, AccessControl) were designed to complete the information infrastructure. The AttributeTable Interface Manager (IM) contains interfaces for accessing information on the four basic attributes, and the RelationTable IM covers interfaces for accessing the six basic relation tables. These IMs deal with EJB entity Beans. The InformationGate IM provides useful interfaces to integrate interfaces of the AttributeTable and RelationTable IMs internally using session Beans. The AttributeInstanceTable IM manages interfaces which access the product attribute instances from the ontology through the DAML query engine that was implemented using XPath. And, the AccessControl IM provides interfaces to get/set access rights according to a role-based access control policy, which is useful on the Internet.

There are five classifications of roles and objects: administrator, DB administrator, trusted company, affiliated company, and guest user. Any query from a user to access the final target objects of a DB table, via a component of the InformationGate or the AttributeInstanceGate, should obtain permission through authentication by a requestor's certificate. The Web service environment will be added to this information infrastructure with server-side components in order to provide the user and automating agents with semantic service descriptions and interoperable interfaces on the Web to access the design attributes.

4.2 Web service environment

Since the Web Service Description Language (WSDL) and Universal Data Description Interface (UDDI) of Web services do not support higher levels of service such as automatic discovery, composition and interoperation, and execution with semantic description, some service description frameworks such DAML-S [6], Business Process Execution Language for Web Service [7] (BPEL4WS), and Web Service Choreography Interface [8] (WSCI) were proposed. As DAML-S is based on strong formal semantics and provides well-defined semantics for automated discovery, invocation, composition, and execution monitoring for Web services compared to that of other frameworks, we can utilize the advantage of semantic activities from DAML-S fully for future agent applications.[5] The Web service environment presented in this paper consists of a service description in DAML-S at a higher level and Web service as its service grounding. Although automatic environment for composition, execution, and discovery in DAML-S is not matured enough, we selected this language framework for future composition and discovery for agents.

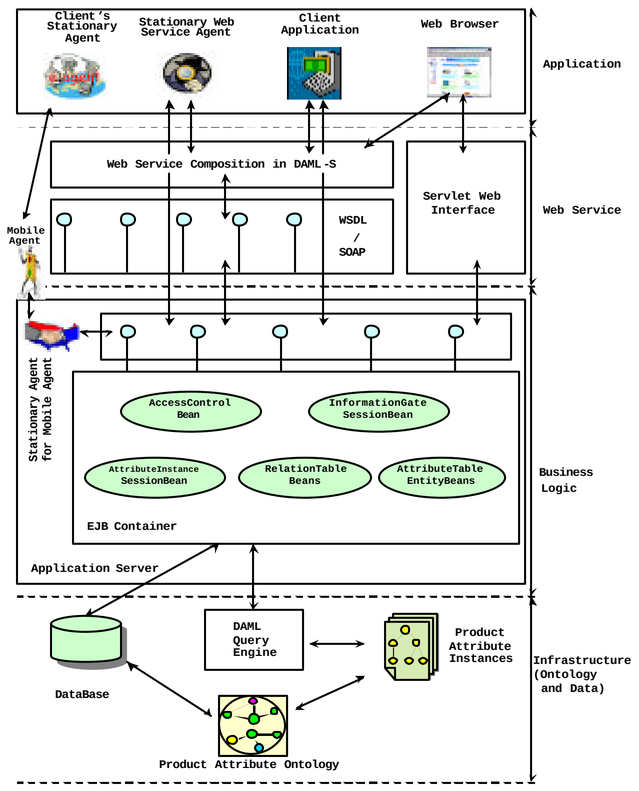
4.2.1 Service description in DAML-S

In this section, we explain service annotation on the information infrastructure for design support. In the concept of DAML-S, we defined basic and necessary classes to describe the ontology for low-level terms such as output list types and the details for describing each item of product design attribute and relation. The service profile contains not only the service name, the contacts and description, and the actor, but it also contains mainly functionality descriptions to describe the specification of functionality services and conditions such as input, output, precondition, and effect (IOPEs) of the services. Here, there is analogical information in the specification for the interfaces of our software component where the IOPEs which function to access the product design attributes. The IOPEs in the DAML-S were extracted from this information easily.

In the service model, process ontology contains only the atomic processes to provide elementary services or composite services in reference to the relational ontology for product design attributes as mentioned in the previous section. For example, we can extract all qualities and functions related to a component with one service invocation. The domain and range of every interface related to the design attributes were defined also in this process ontology. For the basis of the service grounding to Web services, we used WSDL and SOAP mappings. Of course, a service requester can use RMI/IIOP interface conventions because our information infrastructure was constructed with EJB components.

4.2.2 Web services

To build a Web service for service grounding, we used the wrapping method of Web service construction methodologies because we already have server-side EJB components and well-defined service interfaces. Wrapping the existing EJB component interfaces with Web service interfaces worked well because of their similarities. Since the existing component interfaces have an interface type model, an operation definition, pre- and post- conditions, and exception handling, it is possible to wrap these contents of the EJB component interfaces in the Web service interface directly. When EJB interfaces are wrapped by a Web service interface, initialization processes (i.e, the mechanisms for processing naming services, and finding home interfaces and remote object references for EJB beans) in the wrapper routine should be managed with proper policy because the processing time for this initialization procedure takes much more time than that needed to invoke the methods of EJB beans. In our implementation, we managed this by preparing a special objects cache table. We also used a WSDL description to wrap every interface of the EJB components that access the attribute data in the information infrastructure. Web service implementation employs a Java Web Service Development Pack (JWSDP) and an EJB component J2EE Server. Figure 2 shows the entire architecture of the information infrastructure and the Web services for automating the design attributes.



Client's Stationary Agent
Stationary Web Service Agent
Client Application
Web Browser
Application
Web Service Composition in DAML-S
Servlet Web Interface
WSDL / SOAP
Web Service
Mobile Agent
Stationary Agent for Mobile Agent
AccessControl Bean
InformationGate SessionBean
AttributeInstance SessionBean
RelationTable Beans
AttributeTable EntityBeans
EJB Container
Application Server
Business Logic
DAML Query Engine
Product Attribute Instances
Infrastructure (Ontology and Data)
DataBase
Product Attribute Ontology

Figure 2. Semantic Web service environment for product design support in supply chain.

5. Prototyping Agents

The application-centric agent (we call this Design Support Agent: DSA) with some intelligence to carry out example scenarios on Web services and server-side component interface services is designed. We classified the goal of the agent into four work-types according to simplified problem analysis for design support in SCM.

Work-type 1: Extraction of design attribute information

Work-type 2: References to related design attribute information

Work-type 3: Extraction of effects according to changes of attribute variables as independent variables

Work-type 4: Reasoning about which result set is most suitable for user requirements or constraints

The combination of work-type 1 and 2 can be achieved with primitive methods of the information infrastructure. For example, to find all of the functions related to the component "Filament" of "Light Part", we can use the following methods in turn (in Web services, the corresponding interfaces can be called): GetAll(), GetByComponent("Light"), and GetByComponent-("Miniature Bulb") of Component_Cost_Bean, and GetByComponent("Filament") of Component_Function_Bean.

Work-type 3 can be obtained by investigating the changes in dependent variables within a given domain of the four design attributes according to changes in independent variables in the same domain (see Scenario A).

Finally, for work-type 4, an agent can receive information about detailed design attribute items and user preferences as input. The routine to prepare for analysis by the agent requests detailed data from the interfaces of EJB components or the Web service, and obtains the data to be used for the agent's reasoning. The reasoning routine deduces the best candidate combination of target design attributes by forward-chaining based on user preferences as well as data obtained in the previous step. Example questions for this case were illustrated in Scenario B.

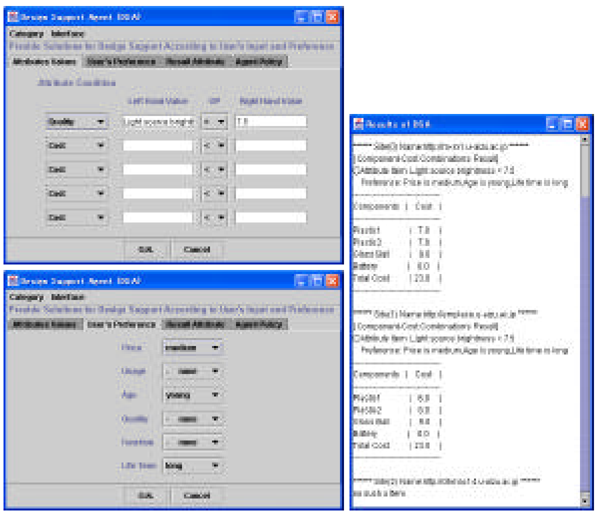
The general sequence of the agents in this research is first to browse the target attribute, second to input the user's request (attributes, preferences, target attribute, and agent policy option), and then to deliver the result after the agent's analysis. A simple Web demonstration of DSA with examples is available at http://emplace.u-aizu.ac.jp/DSASCM/DSADemo/.

5.1 Agents on Web service

Server-side EJB components and Web services for design attributes provide a suitable environment for application-centric agents to work efficiently because they have well-formed standard interfaces from the software component.

The full power of semantic Web service technology leverages more robust infrastructure for autonomous and intelligent agents [9]. Also, more mature usage of agent oriented software methodology will contribute to the architecture for more reasonable agents and environments with autonomy , intelligence, and collaboration.

We have built a stationary agent on both the server-side EJB component service and the Web services using Simple Object Access Protocol (SOAP). A user can give a command to a Web service stationary agent located on the client side. This agent processes the services that have been extracted from the service model in DAML-S through fixed steps. The agent selects a suitable routine of several pre-defined solutions to solve the problem according to the work-type statically, obtains necessary design attribute data from Web services, deduces the best candidates, and reports the results to the user. An example of a stationary agent GUI running on the client side with its result screen is shown in Figure 3.



(a) (b)

Figure 3. Screen of agent example. (a) Tap window of design support agent (b) Example showing the results from the agent.

5.2 Evaluation of agent

To investigate the other aspects that need to be considered as well as the evaluations when DSA is deployed in the Web service environment, we compared the agent's response time according to communication protocols (RMI/IIOP of EJB and SOAP).

Also, the efficiency of additional relation ontology is evaluated. To carry out this experiment, three different test information infrastructures were constructed, which have different instances according to each company's SCM but share the same design attribute ontology. We measured the agent's response time from when the client agent issued the command to when it received the results. The response time includes time for the initialization process to get remote objects, time to invoke the remote interface and obtain results, time to retrieve data from the information infrastructure, and time for the agent's reasoning.

Three experimental groups were made as follows: group 1 covers work-type 1-2, group 2 covers work-type 3, and group 3 covers work-type 4. In our experiment, we counted the average number of interface invocations and their initialization processes for each unit step as follows:

Group 1: Invoking interfaces (5), Initialization processes (2)

Group 2: Invoking interfaces (8), Initialization processes (6)

Group 3: Invoking interfaces (24), Initialization processes (3)

The average response time when using SOAP is 10 times greater than that required when using EJB (1.2 seconds) in all groups. As Figure 4 shows, the response time of an agent varies with the loop count, and the number of agents accessing service interfaces simultaneously increased linearly. However, the slope of the EJB case is relatively low (1.1 seconds/loop), but that of SOAP is high (11.5 seconds/loop).

We observed that SOAP needs more response time due to transmission processes (including serialization-deserialization at client/server sides) of SOAP messages and the management of remote EJB objects in wrapping classes of Web services. We found that the initialization process significantly influences the performance of the DSA system, especially via SOAP (Figure 4 (a), Group 2). In addition, as SOAP generates more loads on the server than EJB, the number of invocations of an agent is a more important factor in SOAP (Figure 4(b), Group 3). This tells us that a decrease in the number of initialization processes and the invocation of service interfaces is necessary when designing Web services.

Next, to evaluate the efficiency of service invocation numbers that differ according to additional relation ontology, we compared the invocation of elementary Web services which use separate interfaces, with that which use interfaces with composite Web service functions on additional relation ontology. Suppose an agent mainly uses uniformly mixed composite service invocations based on elementary service invocation, and can find the target attribute data at every target site with uniform distribution. The average number of service invocations of DSA in our experimental environment is described as follows:

$$N_{invoke} = (1 + F_c/2) \cdot N_b \cdot N_{sites}$$

where:

F_c is the composition factor which describes the average number of elementary services to be composed into one composite service. For example, when we use two cross-references of attributes such as component-quality relation, F_c is 2. When there is no relation, or we use a composite service like our environment, F_c is 1. And when we use triple cross-references of attributes, F_c is 3, etc.

N_b is the number of basic invocation steps required to achieve the goal by an agent at a site.

N_{sites} is the number of sites that are searched by an agent.

As mentioned, if we use an additional relation ontology, F_c would become 1; thus, the average number of invocations will always be $1.5 \cdot N_b \cdot N_{sites}$, which is independent on F_c. In other cases, when a composite service is divided into a corresponding number of elementary services (F_c is larger then 1.), the average number of invocations will be $(1 + F_c/2) \cdot N_b \cdot N_{sites}$.

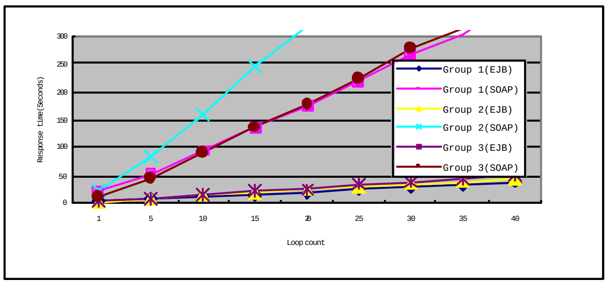
Since the current information infrastructure uses $F_c = 2$, the average increase rate for this case in number of invocations, compared to the case where $F_c = 1$, will be 1/3. If F_c is larger than 2, which means a composite service is divided into a larger number of elementary service calls, the increase rate will increase. Response time will increase abruptly according to the increase in the number of invocations if the response time to unit invocation is large such as the case when using SOAP. Furthermore, if the processing overhead of composition, orchestration, and execution of Web services is considered, the amount of response time will increase more than the corresponding amount of increase in invocation number.

6. Discussion

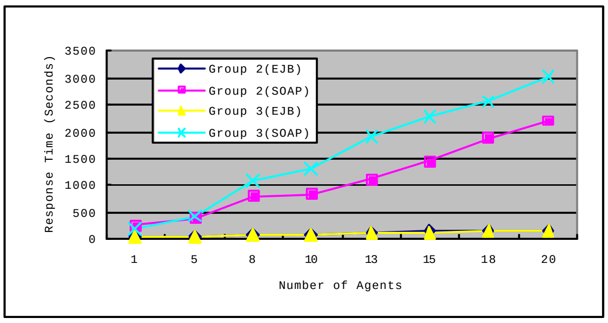
A semantic Web service environment for agents to support the management of product design attributes in a supply chain has been developed taking into account the perspectives of both the industrial needs of management science and the technical needs of computer science. In this research, we found that all of the considerations in the ontology of the design processes, the business logic, the composition and execution of Web services, and agents are needed to achieve better performance, especially in response time, of the agents in the semantic Web service environment.

Considering related work, we can illustrate simple prototype for travel agent to introduce semantic Web service application using Congolog [4]. This work suggested simple scenario for reserving travel schedule on semantic Web service, and introduced simple example of agent's planning in Congolog. The Retsina calendar agent can make travel arrangements for some schedule [10]. This agent is going toward composing several Web services for traveling with automatic Web service discovery. Our work suggests from the bottom to deal with ontological data infrastructure, ontology query engine, and publishing service, to service composition under same semantic Web service framework for design support in SCM.

We continue to extend our information infrastructure to provide higher semantic functions such as semantic querying using an ontology query language. Ongoing research to relate this system to real SCM solutions includes the pragmatic modeling of supply chain graphs for design attributes, granular security processes, and automatic mutual conversion between product design attribute information and the existing SCM information. Building more rational agents to full support design through multi-agents that have improvements on the axes of autonomy, intelligence, mobility, and collaboration which are based on semantic Web services with a robust foundation of rationality is underway.



(a)



(b)

Figure 4. Experiment results of agents. (a) Response time vs. loop count. (b) Response time vs. no. agents.

7. References

[1] M.N. Huhns and L.M. Stephens, Automating supply chains. *IEEE Internet Computing*, vol. 5, no. 4 (July/Aug 2001), pp. 90–93.

[2] I. Paik and W. Park, Software Component Architecture for an Information Infrastructure to Support Innovative Product Design in a Supply Chain. Forthcoming in JOCEC (Journal of Organization Computing and Electronic Commerce), Lawrence Erlbaum Associates, NJ.

http://ebiz.u-aizu.ac.jp/papers/JOCEC/iiswcompo.pdf.

[3] J. Hendler, Agents and the Semantic Web. *IEEE Intelligent Systems*, vol. 16, no. 2 (Mar./Apr. 2001), pp. 30–37.

[4] S.A. McIlraith, T.C. Son, and H. Zeng, "Semantic Web Services", *IEEE Intelligent Systems*, vol. 16, no. 2 (Mar./Apr. 2001), pp. 46–53.

[5] Boualem Benatallah and Quan Z. Sheng, and Marlon Dumas, The Self-Serv Environment for Web Services Composition. IEEE Internet Computing, vol. 7, no. 1 (Jan./Feb., 2003), pp. 40-48.

[6] The DAML Services Coalition, DAML-S: Semantic Markup for Web Services, version 0.9. (April 2003) http://www.daml.org/services/.

[7] T. Andrews et al., "Business Process Execution Language for Web Serivces version 1.1," BEA Systems, IBM, Microsoft, SAP AG, and Siebel Systems (May 2003).

http://www-106.ibm.com/developerworks/library/ws-bpel/.

[8] A. Arkin et al.,"Web Service Choreography Interface (WSCI) 1.0," draft specification, BEA Systems, Intalio, SAP AG, and Sun Microsystems (2002).

http://www.sun.com/software/xml /developers/wsci/wsci-spec-10.pdf.

[9] Massimo Paolucci and Katia Sycara, "Autonomous Semantic Web Services," IEEE Internet Computing, vol. 7, no. 5 (Sep./Oct., 2003), pp. 34-41.

[10] T.R. Payne, R. Singh, and K. Sycara, "Calendar Agents on the Semantic Web", IEEE Intelligent Systems, vol. 17, no. 3, May/June 2002, pp. 84-86