Abstract

A Framework based on Domain Object Interface for Web Services in heterogeneous Distributed Environments

Lim, Eun-Cheon
Department of Multimedia Engineering
The Graduate School
Sunchon National University
(Supervised by Prof. Sim, Choun Bo)

An object oriented software defines many objects and is operated by communication between objects. Object oriented softwares in distributed environment are constructed by web service. It requires searchablity, effectiveness, reliability, stability, and availability for services and accessibility to domain objects and execution performance to implement successfully web services in distributed environment.

In practice, it is hard to realize all of the fundamental architecture to build a web service application so that researches for the development of web services with frameworks has kept going on. Researches for web service framework resolved overall problems that are occurred on registration and search for the web service, communication protocol, orchestration, and execution, but developer should consider much about standard web service technic if he or she manipulate business logic with web services. The problem is occurred when displaying data sources to users in legacy web application after making a connection between a web service framework and existing data sources, namely, the problems are

mapping between domain objects and data sources, connection between local business logic, configuration and deployment of the business logic for web service, and complexity of the execution. It requires developers additional complexity because they cannot use existing development model so that service provider need a lightweight framework that minimizes wide fluctuation of existing development model.

In this paper, we propose a *Meta model* of the web service for devising convenience on searching, configuration, execution and a *Domain Object Interface(DOI)* to handle *Meta model* and design and implement a *Distribution Oriented Knowledge-base of Dynamically Operable Web Servcies(DOKDO-WS)* framework based on the *DOI*. The proposed DOKDO-WS framework offer a web service registry to enroll and search business logics in a domain and a web service browser to identify and execute and a functionality to orchestrate dynamically or statically. Since service is added if that pass the verification process so only executable services should be shown in runtime. Local services is executed by reflection and remote services is executed by the SOAP proxy instances and services that navigated by web service browser is executed by popular web browsers. At the same time, the meta model of the proposed DOKDO-WS framework is defined by XML Schema and is divided into data meta model and service meta model. The service meta model defines local and remote services. The data meta model is used to generate automatically tables, views and stored procedures that have shown high performance in the most persistence operations. Domain objects are transferred in form of the XML by XML transformers and are viewed to user on an appropriate GUI by XSLT processing.

We evaluate performance of the service registry at the factor of

processing time to verify the effectiveness, stability, accessibility of the framework and carry out performance evaluation about the transaction processing in a bank application to evaluate processing power of the DOI. In service registry, average times on enrolling local services, enrolling remote services, modifying services, and removing services takes times on average each 59.0646ms, 40.2767ms, 114.2569ms, and 52.14263ms. Operations on the single service are based on hashing and takes only 0.0012ms on the average. The operation of the enumeration on the total services are increased or decreased in proportion to the number of services, it takes 0.03212ms when there are 10 services registered. In performance evaluation on processing of the domain objects, we performed tests on the factors that are DBMS, ORM framework, the type of persistence operations. In the MS-SQL, Oracle DBMS, the proposed DOI surpasses other frameworks in insert, modify, delete, composite, and select operation, especially at select operation that is frequently used in web service environment the DOI wins an overwhelming result. However, in the MySQL DBMS, get a sluggish performance, which caused by faults on tested JDBC driver. The DOI is faster 5 times than other ORM frameworks in evaluation on the C language. The DOI make 0~3 errors on 100,000 requests. Based on performance evaluation, the DOKDO-WS verify that has higher effectiveness, stability, and reliability as a service registry and an execution engine.

Key Words : web service framework, web service registry, web service orchestration, web service execution engine, domain object, ORM, SOA

I. Introduction

1. Outline

To interact among services as reducing coupling of business logics in distributed environment, several technologies have been repeatedly growing up and falling away. Existing distributed environment technology has developed their own course for transparency of the location and inter-compatibility. However, it is hard to make an environment that is not dedicated to the specific development environment and a user should define an interface on each platform for searching objects, and then the user should pass a message about the object through broker objects. Moreover, domain specific services such as session management, and security problem should be designed and realized in user own way. XML-based standard web service that is implemented for the purpose of distributed environment has appeared to solve those problems.

The web services have drawn up a plan and implemented based on the SOA(Service Oriented Architecture) which is an expansion version of the CBD(Component based Development) methodology, which is realized by distributed environment technologies in the past, on the view of the Service. Fields on the Web Service spin off main research fields such as the registration and search[1-10], the protocol[5, 9, 11-13], the orchestration[14-26], the execution[23-30], and the framework which is offering a research on easy development environment to service providers by integrating particular fields keeps going on.

An web service application performs modeling using a domain model.

Domain objects are used to apply domain models into practical business logic. Domain objects require not only modeling process on the repository to be used but also transformed into other form to communicate with users on the run time so that the web service frameworks is needed to perform research additionally on representation, transmission, modification of domain objects. Service providers cannot help escaping from modeling on domain objects whether the domain model would be replaced to semantic ontology[44-49]. Because the domain model is represented by the XML in standard web service technology, research about mapping between the XML and object is keep going on[49].

Proposed DOKDO-WS framework provides a web service registry which offers functionalities such as dynamical add, modify, and remove web service. Web service can be checked and modified by web service browser and stored based on proposed meta model which resides in main memory for better performance and stored periodically themselves into persistence layer to minimize data loss by service failure. Services stored in a domain object of meta model is verified on time domain object entered in main memory so that user can see only executable services on the run time.

The DOKDO-WS framework supports dynamic service orchestration. Firstly, orchestration in service view layer aggregates functional groups which includes several service units. In contrast, orchestration in service execution layer composes using component aggregation, AOP, generating dynamic proxies. Single or composed operations makes service unit by wrapping. In external execution, every web services can be executed by web browsers and SOAP based proxies. In inner execution, local services can be executed by passing messages to service instances, but remote services executed by generated SOAP proxies. This framework supports

session functionality in part provides methodology to realize web service that never required any changes on existing development environment.

A data model of domain object in business logic is ER(Entity Relation) based. Declared data model automatically generates table and views in supported RDBMS and stored procedure to manipulate this data model. Persistence operations in business logic is performed only by stored procedures which is auto generated or defined by service provider. Service provider can access easily and quickly to persistence layer by declaring minimum configuration and using API.

This paper composes as follows. In chapter 2, we examine carefully existing researches related with registration, search, orchestration, execution, protocol and framework of web service, declaration of the domain object, and mapping to the persistence layer. In chapter 3, describes proposed framework. In chapters 4 and 5, present implementations and performance evaluation, then form a conclusion and open up future works.

2. Motivation

Web service providers need a useful and high-scalability framework. These features can be achieved by supporting methodology that can reuse existing infrastructure of service development and providing compatibility with standard web service technologies.

The framework could check services then dynamically orchestrate and execute. Service registry that provided by the framework could register, search, and modify web services without particular constraints. In general, service providers want to offer only exactly matched services with specific

domain so that dynamically check and modify functionalities for services are mandatory.

The framework should provide interface for XML to easily manipulate protocol. Messages from external of the web services are transferred by HTTP based SOAP whose inner message uses XML based protocol. The service provider can define protocol in the inner part of the SOAP and transfer messages.

It is an important problem to maintain states. When executing web services using existing web browsers, concept of the session is needed. There are no problems to process simple operation such as input, modify and delete with stateless service instances but if the process requires to verify user access privilege on resources which is used on specific services or to maintain association with previous executions, then stateful service instance is necessary.

Finally, the framework should provide method which makes domain modeling easy. Every application needs a domain modeling. Even though domain models are widely spread out, those model in business logic layer is processed in a single service domain. In this process, it is necessary to be automated on mapping among domain object, repository, and presentation language.

Most of web application is storing model in DBMS so writing and modifying SQL is repeated. Domain specific processes cannot be automated so should be developed manually. Repeated persistence operation such as create, read, update, and delete operation can be automatically generated and mapping between model and presentation language can be automated, which decouples greatly business logic and domain model.

II. Related Work

Proposed framework requires wide understanding about the web service and the domain object. For those reasons, we examine problems mentioned in each research in detail, then draw main problems.

1. Web Service

1) Register and Search Web Services

Distributed object technology before web services use brokering and bridging which is centralized method to search and execute. However, in web service based environment, web service repositories not only spread out but also use the Match-making based search algorithms[1]. A process to build web service repository is to construct UDDI[3]. Even though dispersibility of the web service requires also transparency of the location, it needs to divide into Local Services and Remote Services for improving performance of service execution engine. Since local services guarantee higher performance and reliability, important services should be deployed as the Local Services. Because Local Services can be accessed in component level, those don't need to use UDDI based register and search. This occurs degradation of performance instead. Furthermore, shortage of the standard UDDI is very abstract standard, fundamental search functionality, no guarantee on registered service QoS, no service caching, and so on[6].

Web Services searched mainly by IOPE(Input, Output, Precondition, Effect) that is part of the service profile components in OWL-S. In [10],

algorithms and data structures that searches exactly matched services are suggested. Because same word, however, can be translated differently according to domain change, algorithms and data structures which is suitable for semantic web is suggested to infer by changing domain[7, 8, 15]. Web service users not always consider IOPE to search services but sometimes can take additional meta data into account. This information can be added or modified in SOAP message at the time of registration, execution, and changing request on web services[5, 9], which helps framework to search services. However external agent, web crawler or search engine is more effective and practical than embedded search functionality in web service framework[4, 6]. Even though after services completely searched and executed, service provider cannot guarantee any QoS. One of major reasons is that it is filled with many unexecutable remote service if the service registry is static type as time goes by. Remote services should be verified at the time of assigned main memory and applied naturally upon current modifications when business logics are changed[17]. Features that needed to service registries are as follows:

Firstly, meta data should contain descriptions about service. Service providers may not add description on services to be searched if they gain little interest from those services. On the other hand, if they can get much profits from services, they want to expose their explanation in detail to the search engine.

Secondly, service user needs automated verification on non-functional factors. Because most of the service users give higher priority at the factor of high performance, meta data are basically needed to inform users about performance of services.

Thirdly, web services in same domain should have same life cycle with

service container of server. When the container initialized, meta data about services are also initialized to use through domain objects. And service will be unable to search and execute from outside after container or server shut down. This guarantees services always searchable and executable in run time.

Fourthly, endpoints of local services should be generated automatically. This protects web service applications from wrong execution by the root and reduces overhead occurrence while generating and modifying service meta data. Meanwhile, there is no need for secondary repositories and modeling for meta data.

Fifthly, common interface to access service map which can inquire executable services from current domain should be offered. It is mainly responsible for administrator or developer among service consumer to search and compose meta data of the web services in the past. Because for the most normal users who did not access and use directly these services, web service could not gain popularity. So for gaining popularity, easy executable interface is required. Besides, practical requirements for web service registry can be summarized as follows[2]:

- Reuse of the existing infrastructure
- Lightweight approach : use additional complex softwares whether or not
- Complete executable example
- Services belong to specific domain
- User notification : applying modification of dynamic web services
- Integration with other web service registries
- Accurate content : more accurate if the service strongly coupled with service provider

- Control of the content : registry offered directly by web service provider

- easy accessibility and identifiable service description

2) Web Service protocol

Web service can communicate with messages using protocols in application layer. Web service users request messages to and response messages from service provider using protocol such as XML based SOAP over HTTP.

The Axis and .NET framework supports basically the SOAP which requires XML based request and response, it is easy to add additional scheme to application domain[5, 9, 11-13, 50].

In [5, 9], research shows definition of a protocol focused on service search functionality. In [11, 12], researchers define WS-Eventing based protocol which follows ECA(Event-Condition-Action) concept to communicate among devices and design and realize a framework which uses this protocol. If realtime application is needed, protocol can be defined in network layer[13].

One of the problems about definition of the protocol is that existing web browser allows HTTP based request and response but cannot perform directly SOAP request and it is hard to request with adding SOAP message to HTTP based request body. This is the main reason why service instance cannot respond to requester in legacy web environment even though the instance has been generated.

There is a problem about the serialization and deserialization, the marshalling and unmarshalling for a long time that data types used in web service protocol should define manually. XML schema is highly scalable but primitive data structure is different in each framework. Newer protocol

should be defined as that extends the SOAP and is available to request and response upon legacy HTTP.

3) Web service orchestration

After the web service is searched, orchestration process is needed to recreate service unit into valuable service. The service orchestration has proposed by dynamic and static method. The dynamic method can add or modify services in run time. The static method can be mainly divided into two parts. The former reads about operations from static declaration then declare composite component before searching to expose interface from this component. The latter composes pre-defined operations into new structures of the service model. The static method has higher performance but can drop the service availability and flexibility. On the other hand, the dynamic method has lower performance than the static method but it has higher availability and equal performance if there are caches available to use. In that reason, the static method is better if service is local service. On the contrary, the dynamic method is better with remote service.

Existing researches show how to compose and execute web services using standard language such as WS-BPEL, WS-CDL[14, 21, 23]. In [14], they try to map between meta model of the framework and BPEL standard. In [16], they show an implementation about functional change of the web service through the AOP(Aspect Oriented Programming). These are needed to verify service at the step of service binding[18]. In[19], services composed using genetic algorithm based on followings:

- Execution Price : a service requester must pay for the operation.
- Execution duration : sum of processing time and transmission time
- Reliability : technical measurement, related to the hardware and/or software configuration of web services and the network connections

between requesters and providers.

- Availability : probability that the service is accessible.
- Reputation : system's trustworthiness mainly depends on the end user's experience.

In [17], It is error-prone to bind services when using only WS-BPEL so that data structure in verifying services such as CPN(Colored Petri Net), compatibility verification is proposed then research for mapping to standard or dedicated language is progressed to automate[17, 20-22]. Moreover, WS-BPEL, WSCI can be adopted to compose web services.

It is very important to verify semantic whether service exists or do not. In many service orchestration algorithm. Because the verification process guarantees right operation, it can also offer stability and availability of the services. This verification should be performed during initializing meta data and before use some services so it keep away degradation of the performance.

4) Web service execution

The web services can be executed without sharp drop in performance rather than existing distributed technology[51]. A web service execution engine generally creates a service instance per request and destroys or caches it after the execution. In [23], a tool to execute web service from WSDL after composing is proposed. In [26], an example of the network management use Java-based library of the web service which follows standard is suggested. In [27], an example of the mail server and client uses .NET framework which requests web services by asynchronous method is realized.

If the execution engine needs an interaction with user during an execution, it causes selective problem on how to create the user context

and obtain user parameters. Furthermore, interaction with users are not discontinuous so the context that connects current operation and next should be stored in the specific persistence layer. For that reason, the engine needs an identification method that to keep transaction between operations even though service instances were destroyed, which needs concept of the session. Most client proxies of the web services that uses the SOAP can use session ID attached in HTTP header[28]. In [24, 25], a method shows how two-way interaction can be used in session concept. It processes transaction with session and TARGET performs as proxy through the firewall and NAT.

If a session keeps in several operations, a service instance can receive security services from external context. In [29], an algorithm is proposed which verifies access privileges to the resources when a behavior executes that is described by declaration of the choreography. The W3C suggests specification such as WS-CDL and WSCI. In [30], it suggests a XACML(eXtensible Access Control Markup Language) based management server of the web service to integrate functionality of the XML based message encryption and digital signature and then extends SAML(Security Assertion Markup Language) to verify delegation which is passed to service provider. In addition, the execution engine should offer an easy developing method, definition of the primitive data type, serializer of the domain object, and marshaller of execution context.

5) Framework

The purpose of the framework is that provides architecture to integrate overall functionality about register, search, protocol, orchestration, execution, and verification of the service. It consumes much time and cost if a developer of the business logic implements whole concept of the web

service.

The dynamic web service composition, which functionality of the web service is described by the WSDL and the WSCI expresses the interaction among web services, uses the Petri-Net to verify web services[31]. The ISCF organizes itself a distributed web service architecture which uses OWL-S in the business logic layer and the data access layer, which describes data model by ISCFTasks[32]. The SASO involves user, compose, execute, and service layer and the user layer contains semantic modeling, service query model, service deployer, and service executor, which uses ontology derived from WORDNET to explain relations between concept and concept. Execution of the service is performed by the BPEL4j library[33]. In Bottom-Up approach, it selects out a problem occurred in the BPWS4J engine, which processes the BPEL4WS, as static method on the service composition. Because it cannot chooses service through automatic reasoning so it generates knowledge base by service profile component of the DAML-S where service can be searched by the DQL(DAML Query Language), which reduces cost of the service composition when statically composes. In theory, because the composition tries to automatically compose though service providers are unknown about functional factors, a fully implemented semantic is needed. However, because fully implemented semantic that applies various alteration in society and culture and comprehension about whole domain between service provider and users isn't everywhere, existing approaches can be applied only experimental services except mission critical services[34]. In the METEOR-S framework, a service represented to the OWL by the Constraint Representation Module and estimate service dependencies, querying and cost by the Cost Estimation Module. At this time, it

estimates cost for procurement, delivery time, compatibility with other suppliers, relationship with supplier, reliability of the supplier's service, and response time of the supplier's service as the cost of the service itself. And the user has to specify aggregation operators for QoS parameter such as the aggregation about single process execution time, cost of invoking all the services in the process, cumulative reliability, cumulative availability, and domain specific QoS. The constraint optimizer optimizes based on whole constraints. The runtime module converts the abstract process and service templates to an executable BPEL which is executed by BPWS4J engine[35]. In the template-based approach, single composed process is represented using template. It introduce abstract process type to extend the OWL-S. By composing declaration of abstract processes and preferences through the Perform statement, it tries to match and rank services. The HTN-DL is one of the template-based approach. To remove a weak point from the HTN planning, it defines extended HTN-DL[36]. A Java and LISP based framework, IRS-III(Internet Reasoning Service), which can generate WSMO based semantic web service is proposed. It offers a light weight approach to deploy semantic web services and executes suitable web service for user demand. A web service is declared by extending goal of the WSMO. The IRS-III provides browser, WSMO editor, deployment client, and execution client. Those tools can compose statically, services cannot keep session among executions[37, 38]. The SHOP2 converts process model and composes process of the OWL-S into dedicated domain model and planning of the SHOP2 and then executes plans. The SHOP2 is a HTN planning system and operates based on the AI planning. The planning on Golog-like languages that is based on situation calculus is performed offline. It indicates that the planning starts after information-

providing Web services to simulate in state of correctly matched between planning and execution since world can be changed during planning and execution. Furthermore, an execution example shows that the SHOP2 Planner generates plans based on personal schedule and then executes those[39]. The SWORD provides static composition and web server as independent from the standard technology such as SOAP, WSDL, UDDI, RDF, DAML. It can plan for specific service based on rules which are often used in logical language such as the Prolog and then inquires services by rules. Plans are stored based on ER model in form of XML so that input and output of the service can be organized more than one entity[40]. The Plaengine framework composes services based on cases which have service states of the initialization, goal, and progress in form of the logical expression. The interceptor can handle much situations among service execution at before execution, after execution, exception, and recovery. If the service execution fails, then it tries recomposition. Thus, the stability and availability is moderately high[41]. In the SWS compares many frameworks which looked around before in this paper[42]. The web service framework which uses UML and OWL-S generate manually service profile from description of use cases then search services using algorithm of the dynamic evolution. After that, those services are bound and executed[43].

2. Domain object

Web service is implemented in a domain[45] which makes it unavoidable to do data modeling in specific domain. A web based service application creates schema in DBMS after data modeling[48]. The persistence code to

manage the SQL statements which is built from schema is repeatedly written. However, this repeated work is apt to produce lower system structure in productivity and performance. A web service framework should cut down those work and provide decoupling between objects and the DBMS[38, 44]. Furthermore, data modeling also requires simple and duplicated process such as generation of the meta model, create, read, update, and delete of data. To solve these problems ORM frameworks are suggested. The iBatis performs as a SQL Map and the Hibernate replace writing SQL statement to OO modeling. The .NET framework supports the DataFillAdapter which can generate XML schema from DataSource and fill the DataSet with data values. Those ORM frameworks would not expect and process even complex operations for the specific model so complex persistence operation should be implemented manually.

A framework needs to perform mapping between persistence layer and the XML instead of objects defined by classes which are one of the largest causes for decreasing productivity since it performs only data transfer object(DTO). Because basic unit of the message is the XML in the web service environment, object state should be represented as an XML document, which needs an interface for the domain object.

In [48], data represented by the XML are mapped to class of the object oriented language. Firstly, the XElement method is similar to the DOM when managing the XML but user should register a class for specific XML elements to the handler object which is used to parse XML document. When the handler object parses a specific element, registered class is going to be initialized then values are assigned into that object. Objects that will be registered to the handler object must extend the XElement class.

Secondly, the NaturalXML method attaches meta data to each class. Meta data express relationship between a XML and a field of class. This method has a defect that elements are out of order. Data can be transferred to the XIR which supports Base64 encoding but this form is more tedious and complex than the XML or the JSON.

A domain object is an instance which represents model and view used in services. The domain object can be declared in variable form from the view layer to the persistence layer, which it can be handled if transformed into appropriate form in specific layer when communicating among layers. Every domain object would not keep XML form, if XML form is maintained by force that causes problem on performance. However, the domain object used in web service framework should maintain XML form to transfer messages fluently rather than to gain more performance. So it is fundamental for the functionality to transform from the primitive type to Object into XML form.

In [45], the Naked Objects framework is proposed. This framework can dynamically register service and use it. If services are modified, then a GUI and persistence layer reflects changes so service modeling is performed automatically. In [46-48], an idea to separate service and data code is suggested.

III. DOKDO-WS Framework

1. Entire framework structure

Proposed framework is made up of four layer as like Fig. 1. Layers are partitioned based on the MVC pattern. The view layer offers UIs to normal service user and the execution layer takes charge of processing flows and transmitting data. The service and data model layer provides a domain model which is employed in entire system.

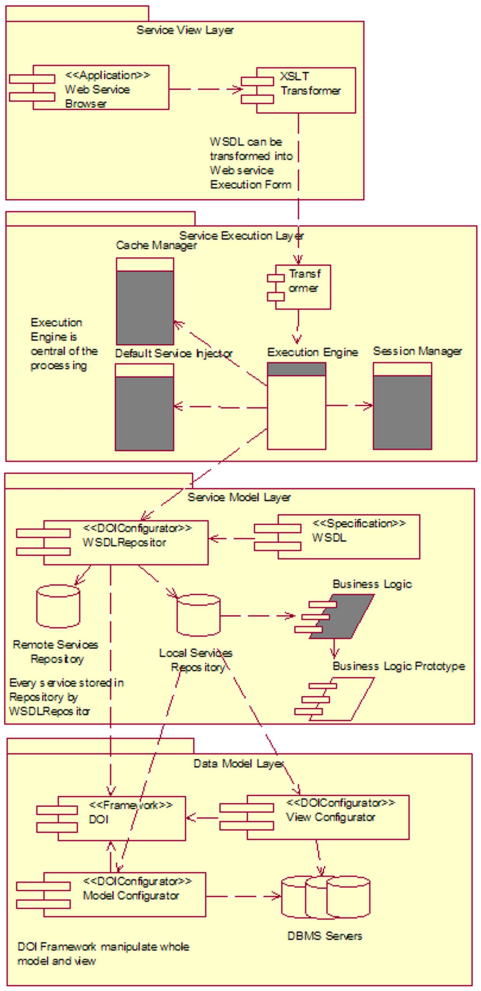

Fig 1. Entire framework structure(diagram)

1) Data model layer

This layer provides a service which handles modeling for data used in the business logic and for views offered to service users. Most of

operations are delegated to the Domain Object Interface(DOI) framework which has an interface for loading, searching, modifying, and storing about XML based domain object. By using this interface, a service provider configures declaration of the domain object model and view on services and the DOKDO-WS framework uses the DOI to generate domain model of the business logic.

2) Service model layer

This layer accesses remote document of the WSDL and then services would be verified. If it was successfully verified, then services registered in runtime repository. For local services, prototype of the business logic that is obtained from components of the business logic makes WSDL objects at initializing the framework. Because the framework only register verified service to the runtime repository, service instances surely guarantees the execution.

3) Service execution layer

This layer decides policy and strategy on how to manage them when request for execution arrives. That type of execution is optional such as synchronized or asynchronized type, whether session maintains or not, the cache policy, and an instantiating method. There are three service at subordinate position of the service execution engine, which is the cache manager, session manager, and service injector.

(1) Cache Manager

Since some operations of the service can be called frequently in short time, instantiating an instance per call causes the performance of the service to degrade greatly so the cache manager caches instances within a given period if the service has been often initialized.

(2) Session Manager

Concept of the session could be needed on specifically composed services. Data which needs to keep their state could be chosen in the service declaration time. If the session is enabled, the session manager must restore previous session value before calling and storing changed value into session repository again.

(3) Service Injector

A service injector injects fundamental services into a web service instances. This service makes feasible to compose dynamically services in the execution layer. Basically injected service is the session manager, which attaches functionality for the session into service instances.

4) Service view Layer

Domain objects in the service view layer take XML based messages. However, a web service provider generally processes results from the service execution layer to GUI. For this, this framework uses the XSLT to translate into a view model which user wanted to. A browsing service and executing browsed operation service which the service providers can use is also provided.

2. Design of the framework

1) Use case

Fig. 2 shows a usecase of the service providing developer whose role is that defines declaration of the model and view for services will be provided and realizes controller classes which represent the business logic

based on this declaration. After that, declaration for controller classes should be registered in a service model. When controller class develops, choose whether expose each operations of the service or whole service and whether use session or not.

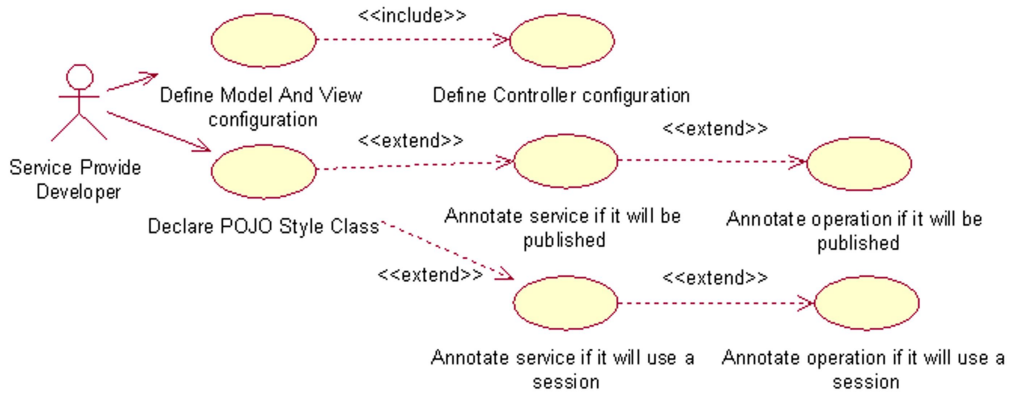

Fig 2. Use case of the Service providing developer

Fig. 3 shows a usecase of the service assembling developer. A service assembling developer checks operations from meta service and then generates proxies based on the WSDL. Generated proxies composes newly operations into valuable services.

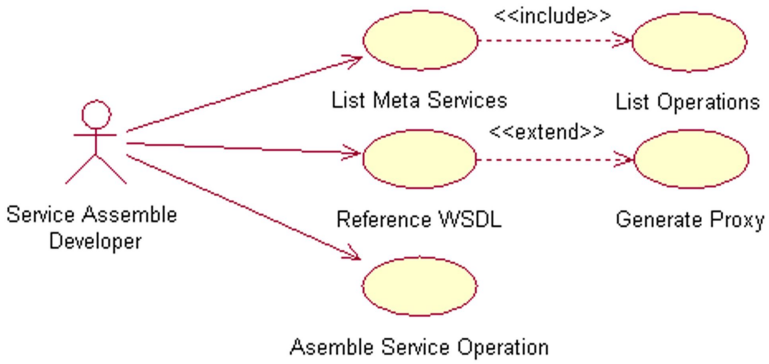

Fig 3. Use case of the Service assembling developer

Fig. 4 shows a usecase of the DOKDO-WS framework. The DOKDO-WS framework generates service models from service

configuration which service providing developer and assembling developer were defined after verifying each local and remote services. This process includes two sub-processes of the generating WSDL objects and service models. After that it generates model and views in RDBMS of the persistence layer. Stored procedures are also generated at the same time which is employed as processing logic for persistence such as create, read for only searchable columns, update, and delete.

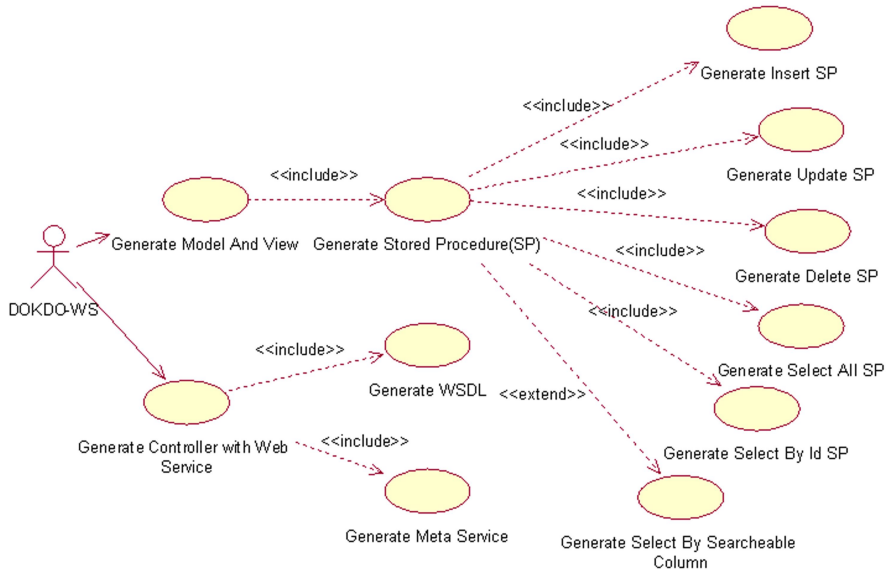

Fig 4. Use case of the DOKDO-WS framework

2) State diagram

Fig. 5. shows a state diagram for initialization of the container. In the first state, the Context Initalize, other framework can be initialized. At next, in the Generate Controller state, the execution engine will be initialized. In the Load Services state, declaration of the service is transformed into the service model. In the Generate WSDL state, WSDL objects are created in each service model. Next state is decided by configuration of the service model, which is either the Generate Model and view or the Context Initialized state which is final state of this diagram.

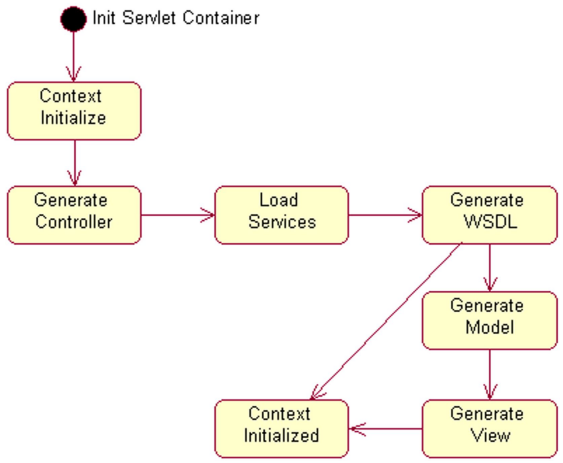

Fig 5. State diagram for initialization of the container

Fig 6. shows a state diagram for execution of the web services. External client would be in the List Services state after context initialized through the state transition in the Fig. 5. Services are displayed in groups by their own namespaces. In the WSDL Queryable state, WSDL document can be obtained from WSDL objects. In the List Operations state, the client can enumerate available local and remote operations. In the next state, operations are filled with parameters. After then, the execution engine invokes services. In the final state, client shows the invocation results from the web service.

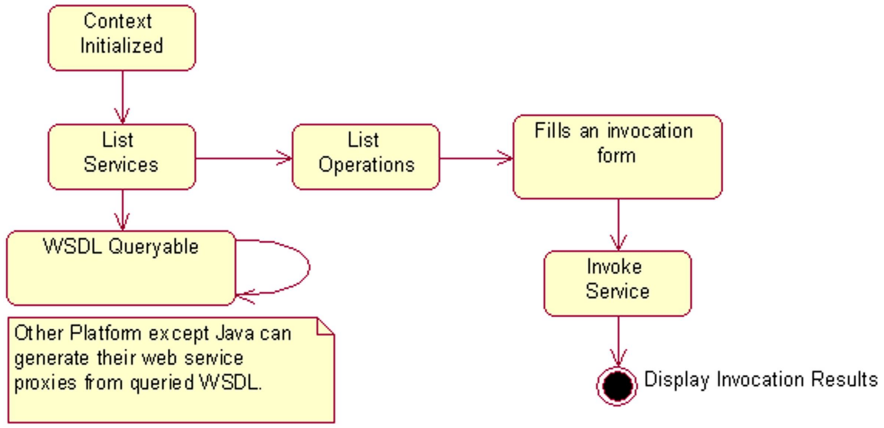

Fig 6. State diagram for execution of the web services

3) Design of the component

(1) Design of the business logic layer

① DOI

Fig. 7 is a class diagram for the DOI. The IDomainObject interface is for whole domain objects which extends IDOMAccessor and IDataAccess Object interfaces. The IDOMAccessor interface is an interface for managing all XML based domain objects from configuration to execution. The IDataAccessObject interface is an interface for loading, modifying, and removing domain objects in persistence layer. A default implementation of the IDomainObject is offered by the AbDomainObject which is an abstract class and initializes the Document object for handling current state of the domain object by the DOM and builds the XPath object for searching Nodes from XML document.

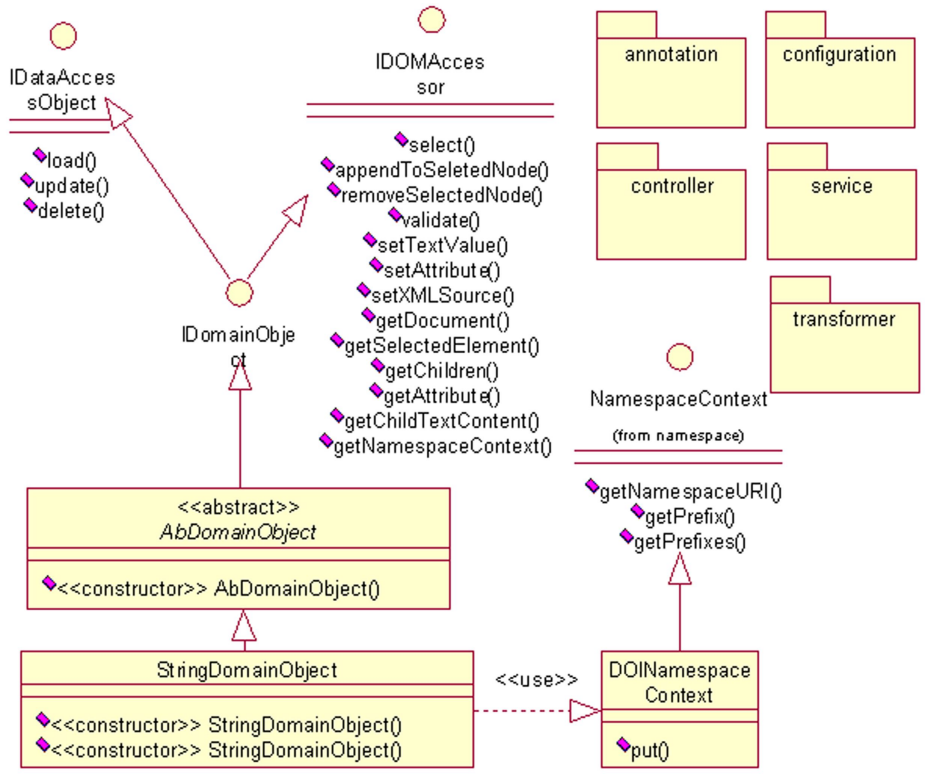

Fig 7. Class diagram for the DOI

② Database Facade

Fig. 8 is a class diagram for the database facade which has a role to manage access to the DBMS. The IDatabaseFacade interface encapsulates functionality which the Database Access Object(DAO) has to be implemented. The ICloseable interface means some child objects would be closed but user of those objects don't need to call this method because this method will be invoked automatically by implementing the Template Method pattern which makes the DBMS connection closed unnoticeably. The AbDatabaseFacade class is an abstract class which provides basic implement for the IDatabase interface. The AbJDBCDatabaseFacade abstract class is an abstract class which provides fundamental functionality for the objects which employs especially the JDBC of the Java language among many connection methods of the DBMS. For the convenience, this framework provides implemented classes for the Oracle, MS-SQL, and MySQL DBMS which has been often used.

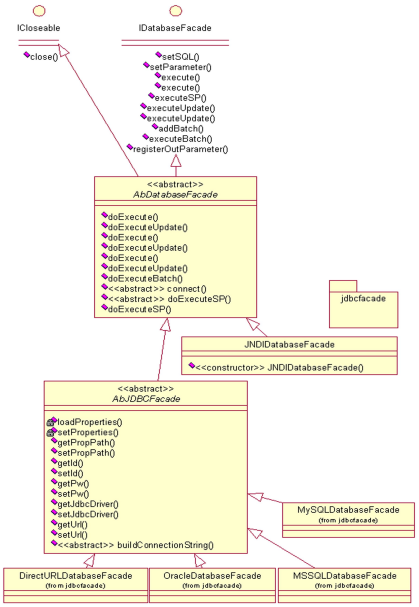

Fig 8. Class diagram for the database facade

③ Configurator

Fig. 9. shows a class diagram for the configurator which reads configuration about the model of the service and data to use in runtime. It defines AbDOIConfigurator that is an abstract class by extending AbDomainObject that implements basic functionality of the DOI to handle all configuration in this framework. For configurating DBMS, it defines an abstract class, AbDatabaseConfigurator, which is divided into two abstract classes for the configurator of the model and view. Finally, concrete configurators are defined which can be created as service instances. Depends on where execution engine, some configurator must be declared and the DOIServletConfigurator class is provided for the Java web environment.

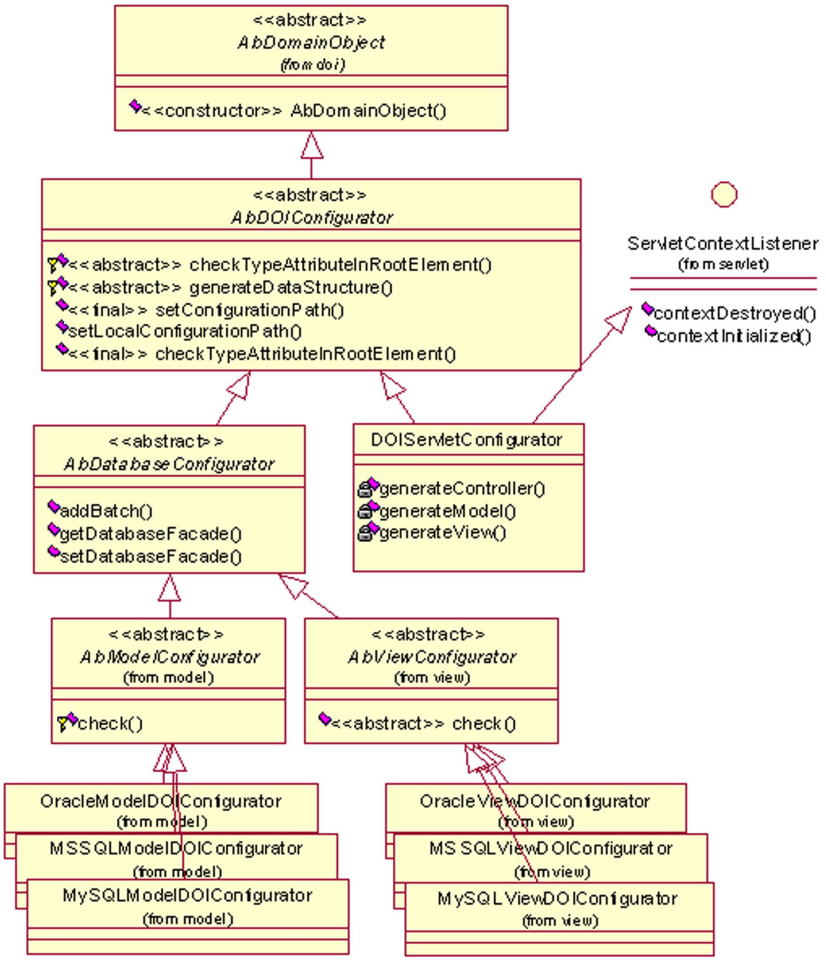

Fig. 9. Class diagram for the configurator

(2) Design of the service meta model layer and the service execution layer

Fig. 10. is a class diagram for repository of the service model and a class diagram executor to manage meta data in there. The service repository provides interface to store or create a WSDL document to execute services at a later time, to enumerate or query services, and to list operations in each service.

The executor has a purpose to execute services and realizes functions of the cache manager, session manager, and service injector that those are

sub programs. The executor to process local and remote services exists separately and all executor should be implemented as derived from DOIXMLExecutor because all data basically has the XML form.

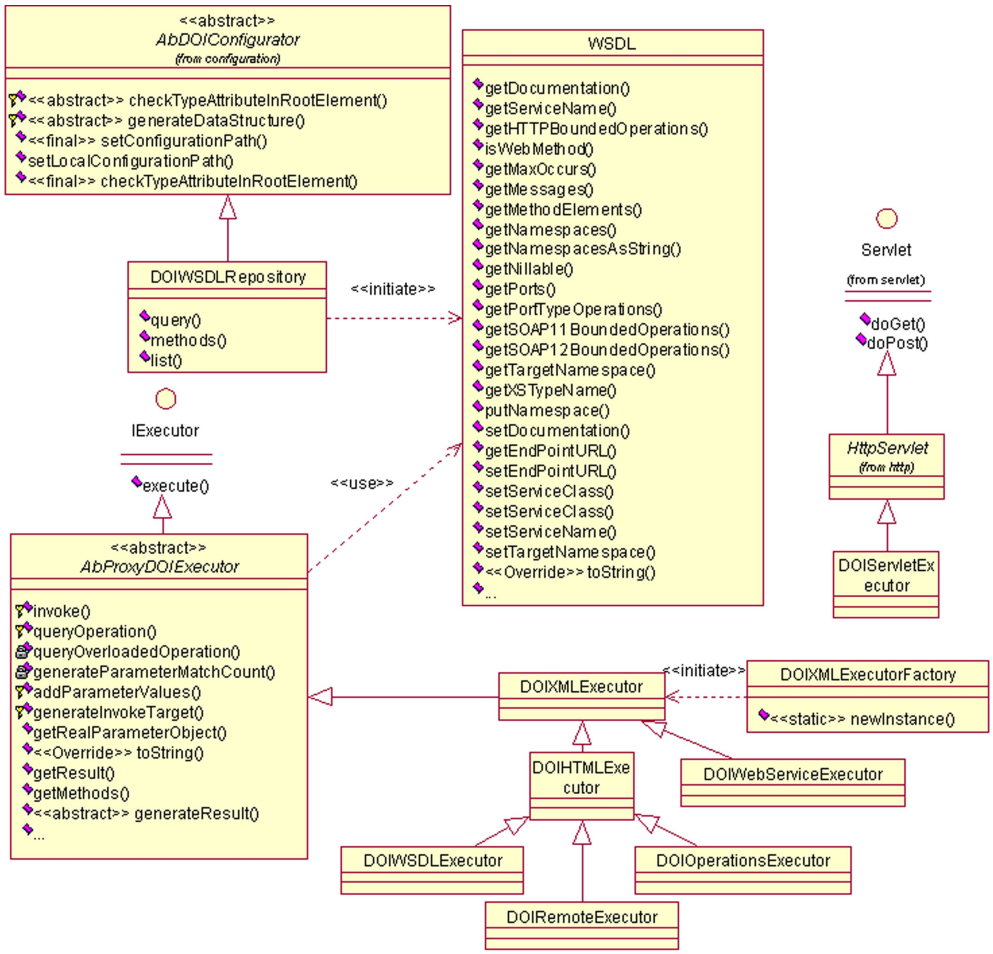

Fig. 10. Class diagram for the WSDL repository and executor

Fig. 11. is a class diagram for the transformer to change the data in the domain object into XML form before passing to the service view layer. When service uses external data from current execution context, the TypeSerializer service converts it into a domain object. In the contrary, when the execution engine passes a domain object to other layer, it will be turned into a string or a form of XML document.

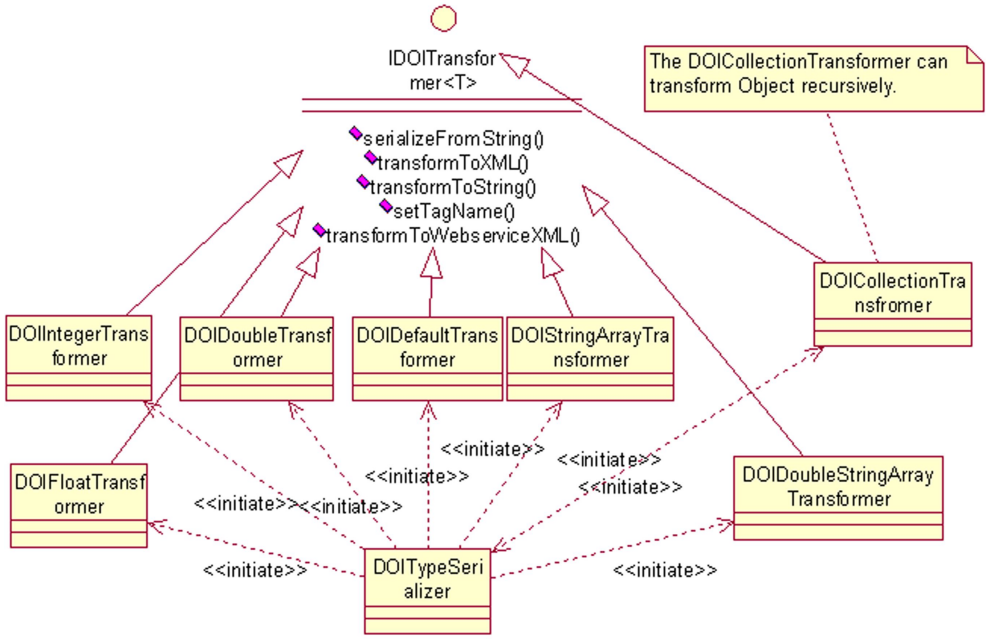

Fig. 11. Class diagram for the transformer

(3) Design of the service view layer

Fig. 12. is a class diagram for the XSLT processor which is used to transform a XML document into other XML document. As using this service, the XML document can be converted into appropriated form to the service view layer. If a string-based stream was simply returned, there is no need to use the XSLT transformation in the view layer but have to be a client application which can recognize the stream.

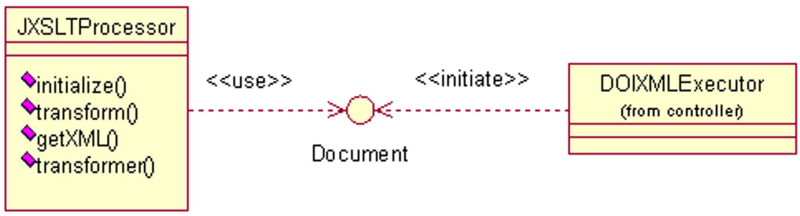

Fig 12. Class diagram for the XSLT processor

4) Design of the meta model

(1) Main meta model

Configurations about every domain object used in the DOKDO-WS framework are originated from the main meta model of Fig. 13. This model is declared as one of the XML Schema. Every meta model instance must have a type value of the specific model in the root node's type attribute, which is to check service providers whether they have known exactly the type of meta model instances and declare them. The modelAndViewGenerate attribute is used to decide the next state of the Generate WSDL state in Fig. 5. If it has the true value, it generates the data model layer. A service provider must declare only one main meta model in a domain because local service repository keeps an entry point for all services and performs hashing based algorithm in searching. In every model, the localPath and remotePath attribute represents input and output path and the id attribute is an identifier for specific model in model declaration.

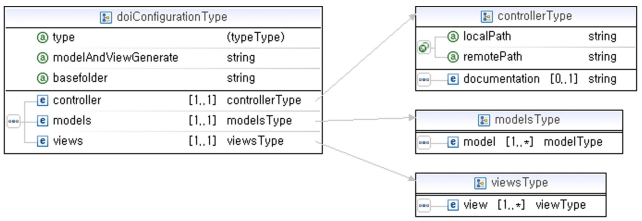

Fig. 13. The main meta model

Fig. 14. shows model for the domain data model and view model. The domain data model and view model has similar declaration. The domain data model expresses all information of the DBMS based persistence which is used in domain of the web service. The dbmsName attribute can apply

only the name of the DBMS which this framework supports. The diName attribute is used to specify a name which is used to look up a service from a directory service by directory interface. Because the reference implementation of the framework uses Java language, so attribute name is jndiName.

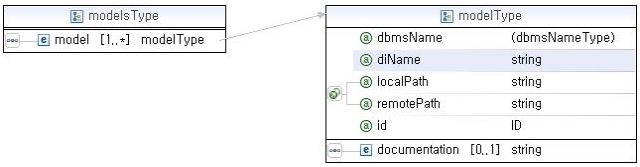

Fig. 14. The domain data model and view model

Fig. 15 is an example of the main meta model.

```
<?xml version="1.0" encoding="UTF-8"?>
<doicfg:doi    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:doi="http://javawide.com/DOIconfiguration" xsi:schemaLocation=
"http://javawide.com/DOIconfigration..." type="configuration" model
AndViewGenerate="true">
<doicfg:controllers>
<doicfg:controller localPath="protocol:///path/filename" />
...
</doicfg:controllers>
<doicfg:models>
<doicfg:model  localPath="protocol:///path/filename"  dbmsName="Some
DBMS" jndiName="jdbc/doi_somedbms" />
...
<doicfg:views>
<doicfg:view  remotePath="protocol:///path/filename"  dbmsName="Some
DBMS" jndiName="jdbc/doi_somedbms" />
...
</doicfg:views>
</doicfg:doi>
```

Fig. 15. An instance of the main meta model

(2) Sub concrete model

① Service model

Fig. 16 shows the service model. The service model is a model to represent a single process which can be thought as the Controller of the MVC pattern. All business logic have equal form without any association whether it is local or remote service. Commonly used nodes are as follows. The id attribute of the controllers element performs as namespace and the name attribute of the controller element is a unique identifier of services. The documentation element describes service and the targetNamespace element means a namespace of the caller.

The serviceClass element is used only in the local service and it means a module name which is employed to create service instance. And the endPointURL means the end-point URL to receive the SOAP request from the outside. Lastly, let's look into elements which are used only in remote service, the wsdlURL is a URL for service description to execute.

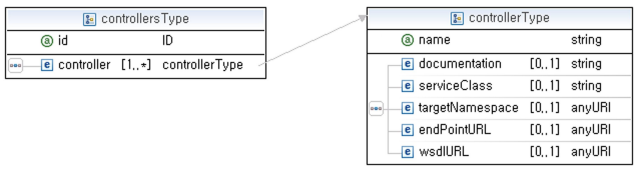

Fig. 16. Concrete service model

② Data model

Fig. 17 shows the concrete data model. The data model is a model to represent object and data that is used in a single process which can be thought as the Model and View of the MVC pattern. The DOKDO-WS framework stores basically data in the RDBMS so the data model has a form of the ER model. The data model has many table or view elements in

a database element. The framework generates the domain data model in DBMS based on the concrete data model. If value of the forceRemove attribute is true, then the model would be forcibly deleted and recreated. The primaryKey element has the same type with the column element but it represents primary key of the ER model. In a declaration, a composite key is not allowed because the configurator service creates internally an artificial key which make a composite key from the internal artificial key and a user defined primary key. Attributes in the columnType is used to specify a constraint, data type, and name when mapping the value from persistence to a domain object.

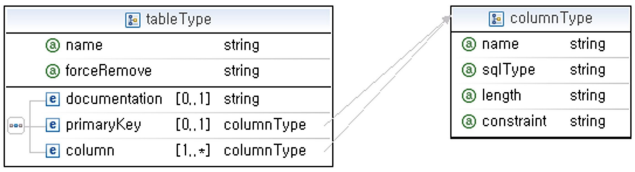

Fig. 17. Concrete data model

Fig. 18 shows the concrete view model which is used in the service view layer. The forceRemove attribute means the view would be forcibly deleted and recreated like the data model. The mapTo attribute can modify a name of data which is exposed to the view layer. The generate attribute specifies the type of a stored procedure what persistence operation is going to be generated. Only the update and delete operation are basically generated for the inner id column but this attribute allows for other columns.

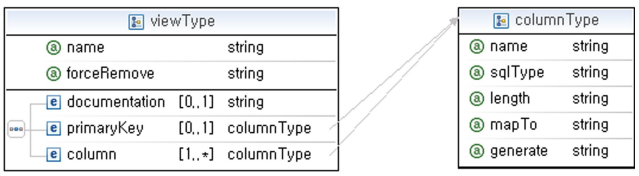

Fig. 18. Concrete view model

IV. Implementation and performance evaluation

1. Environment of the implementation and performance evaluation

In the performance evaluation of this paper, we uses the AMD Athlon 64 bit Dual Core 2.0Ghz for the CPU and the Windows XP Professional SP3 for the OS and the JDK 1.6.0 Update 10 version for the programming environment. The physical memory is 2.75GB and total 3.6 GB in view of the virtual memory. The network is connected in wire and uses the same localhost with the HTTP. We use the HttpClient 4.0 Beta 1 version to request specific service.

The performance evaluation carried out quantitatively and qualitatively on the point of the web service and the domain object. In the quantitative performance evaluation of the web service, the request count is 5,000 and the intensity is about 30 threads/sec. The measure is only the execution time in the server because all other conditions are equal.

In the quantitative performance evaluation of the domain object, the request count is 100,000 and the intensity is about 30 threads/sec with a bank application but the JOXM is always failed with that condition so we adjust the request count to 20,000 and the intensity to 10 threads/sec. Changed factors are type of persistence, operation and ORM framework. The measure is the executed time of the server, received time of the client, and network duration. The network duration is calculated by subtracting executed time of the server from received time of the client. Types of persistence are the JOXM, MS-SQL, Oracle, and MySQL and we

compared their performance on the iBatis and Hibernate. The type of persistence operations are INSERT, UPDATE, DELETE, COMPOSED, READ. The measure for INSERT, UPDATE, DELETE, and COMPOSED operation is execution time of the server and only for the READ operation we use execution time of the server, receive time of the client, and network duration. If execution time of the server is greater than 500ms, it may result when the garbage collection is performed so we exclude that values.

2. Results of the Performance Evaluation

1) Web service side

(1) Qualitative evaluation

Table 1 and 2 compares existing web service framework with the DOKDO-WS. To use standard web service, it employs the UDDI as a repository of meta data. There is a need to own repository of meta data to provide the semantic web service. Because the centralized method for repository inquires descriptions of the service execution from a global service registry so the distributed method is more appropriate to the web service. The lightweight approach means if the framework automates the development process that there is no need for any additional tools or maintaining the existing development method. In the existing framework, only the dynamic web service composition and the UML based evolution framework provides this measure. The language of meta model means compatibility with domain model of the other web service framework. To achieve this, framework would use standard language for domain modeling or provide an interface to handle general domain model. The SWORD

provides such interface. The method of the composition affects largely to performance, convenience, and learning cycle of the framework. The composition of the web service in most of existing framework generates a knowledge base using rules of the logical language then the framework searches and composes services. However it needs more learning time and a mapping tool or API to integrate with standard language. The AI method can compose with minimal configuration but has lower performance. The static method is the easiest and is more flexible but it has a weak point that the service provider should declare all configuration. The execution description is a language that explains services and operations. Using standard WSDL and BPEL has more scalability.

table 1. Comparison with other web service frameworks - I

Items	Dynamic Web Service Composition	ISCF	SASO	Bottom-Up Approach	METEOR-S	Template-based
Storage of Meta data	UDDI	OWL-S Library	SDL	DQL Server	UDDI	HTN
Centralized storage	○	○	×	×	○	○
Light weight approach	○	×	×	×	×	×
Meta modeling Language	WSCI	OWL-S	WORD NET	DAML-S	RosettaNet	OWL-S
Composition method	Rule Base	Rule base	Rule base	Static	Rule base	HTN
Execution Descriptor	BPEL	WSDL/ ICL	BPEL, WSDL	BPEL4WS	BPEL, WSDL	HTN-DL

table 2. Comparison with other web service frameworks - II

Items	IRS-III	SHOP2	SWORD	Plaengine	SWS	UML based Evolution	DOKDO-WS
Storage of Meta data	OCML Library	HTN	Persistent XML Plan	Domain Registry	SWSs registry	Ontology Repository	Domain Registry
Centralized storage	○	○	○	○	×	○	×
Light weight approach	×	×	×	×	×	○	○
Meta modeling Language	WSMO, UPML	SHOP2 Domain(from OWL-S)	Every XML	Own Meta-Model	OWL-S	OWL-S, OWL	Every XML
Composition method	Rule base	HTN	Rule base	AI	AI	AI	Dynamic
Execution Descriptor	WSMO	WSDL	WebL, Filter	WS-BPEL	BPEL4WS	WSDL	WSDL, Function Structure

(2) Quantitative evaluation

0 to 3 errors occurs on average in 100,000 executions of the persistence operation. Fig. 18. shows operations on the service registry of the DOKDO-WS. Operation time of the service registration is increasing as time goes by. This operation takes much time only when service meta data is written to physical disk. The time slope of the local service is steeper than the remote service because the local service verifies existence of the service by trying to instantiate on each added service module. In general, the service modification takes the longest time. Because this operation is composed of two operations which one searches location of the service and another modifies it. The removal operation takes little time as time goes by. In Fig. 19. average time of operations at registration of local services, registration of remote services, modification and removal of

services are each 59.0646ms, 40.2767ms, 114.2569ms, and 52.14263ms. Searching operation of the single service takes only 0.0012ms as average. Enumerating operation is in portion to the number of services. When 10 services have registered, it takes 0.03212ms as average.

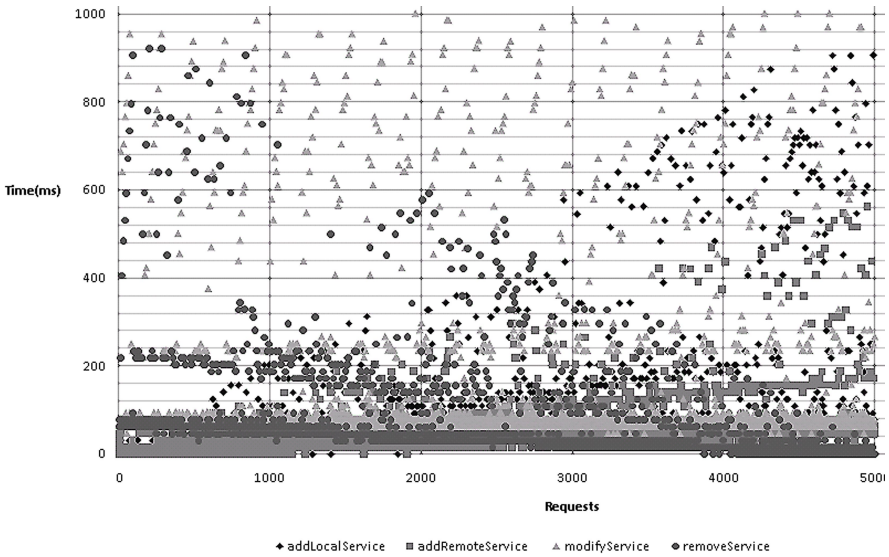

Fig. 19. Operations on the service registry of the DOKDO-WS

2) Domain object side

(1) Qualitative evaluation

Table 3 and 4 compares existing researches related to the domain object with the DOKDO-WS. Most of researches provide their own API about connection and persistence. Storage method is divided into the file system, XML DB, RDBMS. However, it is not realistic to use the file system for persistence layer because processing of concurrency problem or transaction, security is basically required in the web service environment. In the mapping relation, represents how framework restore objects from the persistence layer and save objects to persistence layer. The creation method of the domain object is mainly manual method. Automatic method

raises the productivity. The learning cycle expresses how much it is easy based on the number of tools, concepts, and modules. For example, The Low means that it provides lightweight approach.

Table 3. Comparison with other research related to the domain object - I

Items	Taming XML	Naked Object	Runtime Generation	Decoupling OO Layer	Universal Relation Data Source
Connection Method	API	API	External ORM	API	API
Persistence API	×	×	External ORM	○	○
Persistence Type	File	File	RDBMS	RDBMS	RDBMS
Mapping Relation	Object<->XML or XIR	Properties->Object<->GUI	Object->Dynamic Proxy<->Persistence	DBMS<->Object	DBMS<->Object<->Formatted String
Creation method for Domain Object	Manual(Marshalling)	Automatic	Depend on ORM	Manual	Manual
Learning difficulty	Low	Low	Middle	Low	Low

Table 4. Comparison with other research related to the domain object - II

Items	JOXM	iBatis	Hibernate	DOKDO-WS
Connection Method	API	API	API	API
Persistence API	○	○	○	○
Persistence Type	XML DBMS	RDBMS	RDBMS	RDBMS
Mapping Relation	XMLDBMS<->Object->XML	DBMS<->SQLMap<->Object	DBMS<->E-RMap<->Object	DBMS<->ModelMap<->Object
Creation method for Domain Object	Manual(XPath)	Manual	Manual	Semi-Automatic
Learning difficulty	Low	Low	High	Low

(2) Quantitative evaluation

① INSERT, UPDATE, DELETE, COMPOSED operation

Fig. 20. shows performance evaluation of the INSERT, UPDATE, DELETE, and COMPOSED operations in the MS-SQL DBMS and the JOXM. The registerCustomer operation inserts a row of the 9 columns and the registerAccount inserts a row of the 5 columns. The JOXM is influenced enormously by the number of columns. The modifyCustomer which is the UPDATE operation is 4 times lower in the JOXM and 6 times in the others than INSERT operation. The unregisterCustomer which is the DELETE operation is 4 times lower than the INSERT operation. The deposit and withdrawal which were the COMPOSED operation includes addition, and subtraction then inserts the variation into another table. In this time, regardless of the type of the additional operation, the performance of the COMPOSED operation is 3.5 times lower in the JOXM and 11 times in others than the INSERT operation. As a result, in the MS-SQL DBMS, the DOI framework which is employed in proposed framework has the highest performance, and the iBatis, Hibernate, JOXM by turns.

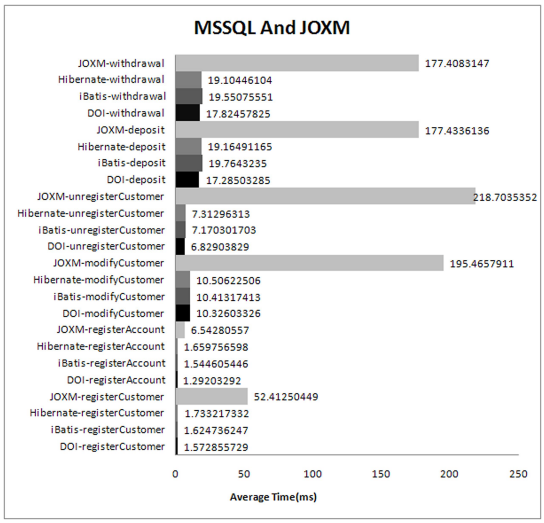

Fig. 20. INSERT, UPDATE, DELETE operations in the MS-SQL and JOXM

Fig. 20. shows performance evaluation of the INSERT, UPDATE, DELETE, and COMPOSED operations in the MySQL DBMS. It performs the best the DOI on the INSERT and DELETE operations and the Hibernate on the UPDATE operation, and the iBatis on the COMPOSED operation. The DOI shows lowest performance on the COMPOSED operation but it shows generally high performance.

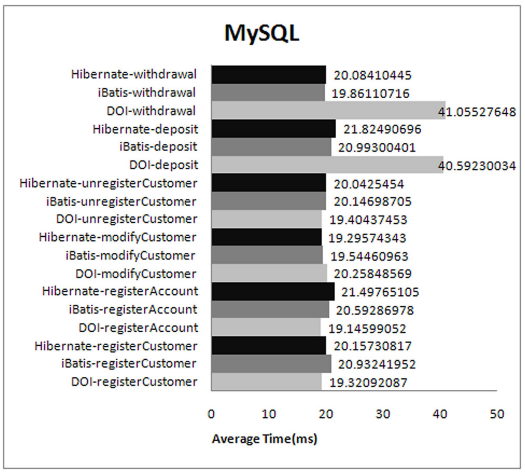

Fig. 21. INSERT, UPDATE, DELETE operations in the MySQL

Fig. 20. shows performance evaluation of the INSERT, UPDATE, DELETE, and COMPOSED operations in the Oracle DBMS. The iBatis performs the best on the INSERT, UPDATE, DELETE operations but the DOI has higher performance on the COMPOSED operation. Especially, on the withdrawal operation, the DOI performs 3 times higher than the Hibernate and 1.3 times than the iBatis.

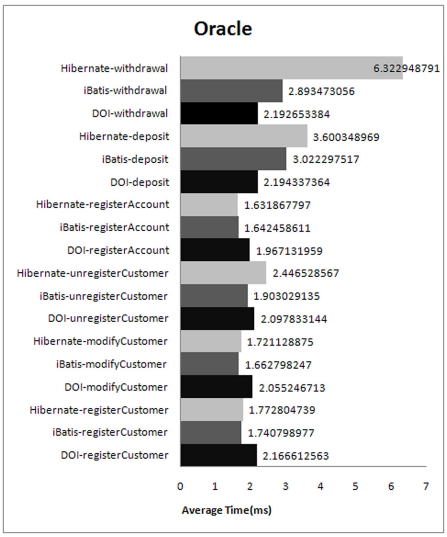

Fig 22. INSERT, UPDATE, DELETE operations in the Oracle

② SELECT operation(Point query, Ranged query)

The SELECT operation is tested separately as the ranged query which fetches 100 rows and point query which fetches a row associated with specific id value on the Customer table. Fig. 23. shows tests on the SELECT operation in MS-SQL. Operations ends with 'client' is receive time of the client. On the point query, the DOI has about 52 times higher performance than the iBatis or the Hibernate. The network duration is similar in most of the framework but the iBatis is 2 times slower than the others on the ranged query. On the ranged query, the DOI has 1.2 times higher performance than the iBatis and 2 times than the Hibernate.

Eventually, on the SELECT operation in the MS-SQL, the DOI shows the highest performance.

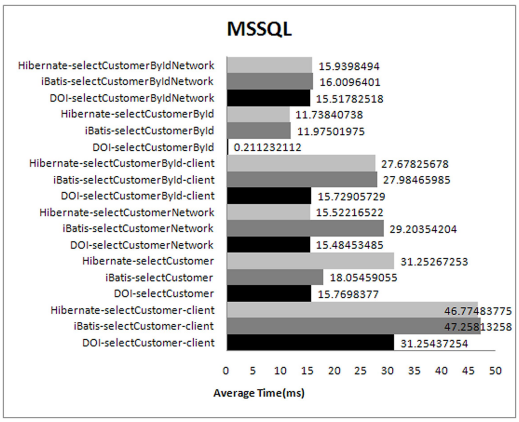

Fig. 23. SELECT operation in the MS-SQL

Fig. 24. shows the performance evaluation on SELECT operation in the MySQL. By the specific faults of the JDBC driver, the DOI shows worst performance because another frameworks fetches result set from the memory but the DOI fetches from the physical disk. However, if this operation evaluates in the C language, the DOI shows 5 times higher performance than another ORM.

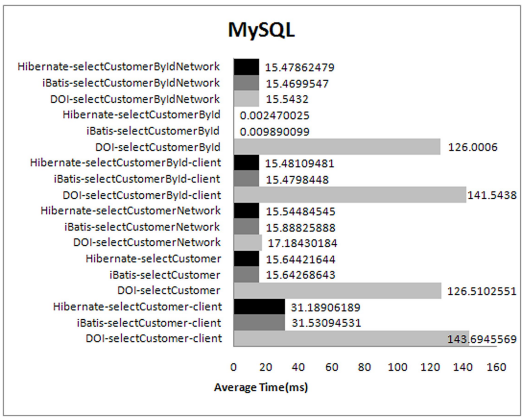

Fig. 24. SELECT operation in the MySQL

Fig. 25 shows the performance evaluation on SELECT operation in the Oracle. On the point query, the Hibernate and iBatis seems even better but

it's little difference around 0.019ms. On the other hand, the DOI is 82 times better than the others on the ranged query and the time difference is about 15 ms. As a result, we can know the DOI shows higher performance than any other frameworks.

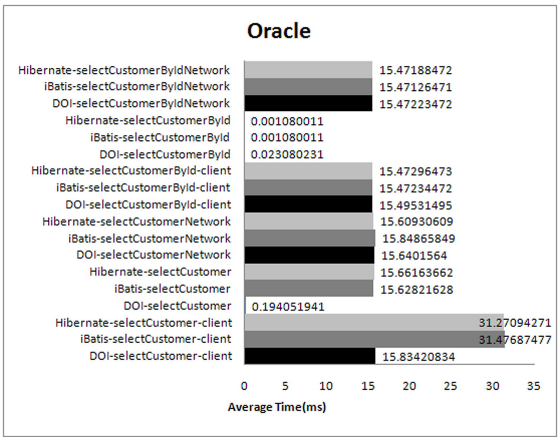

Fig. 25. SELECT operation in the Oracle

3. Implementation

1) Lightweight service repository

The DOKDO-WS framework satisfies most requirement for the service repository that was mentioned in [2]. The binary size is 128Kbytes and has faster initialization. Components which is provided as the API can be used without modification and services on existing infrastructure are no need to change and can be reused.

Service providers can register, modify, and remove services through normal web browser and the consumers can identify instantly the modifications. Because some other service registry can be offered as a service, it's easy to integrate with any other web service registry. Moreover, since the service provider directly affords the web service

registry, it is difficult to be served unsuitable services. Local services depend on the specific service in the current domain and the service provider can serve remote services in the equal domain.

2) Domain Object mapping

It links among the domain objects, the DBMS and the XML through a simple meta model. A primitive data type defined and it performs the serialization or deserialization of data objects and the marshalling or unmarshalling of service instances. The type conversion of domain objects are fulfilled by the transformer service. It also automatically generate operations to access easily to the persistence layer. Complex persistence operations can be registered manually by the service provider.

3) Dynamic composition

Services can be dynamically composed in the service execution layer and view layer. Fig. 26. shows an composition algorithm in the service execution layer. In the service execution layer, a dynamic proxy which has been added execution code before and after for operations going to be execute would provide as a service instance. The dynamic proxy is generated by injection using the Reflection or modification of the byte code using the AOP.

```
Let S = {B, A}, D = {s0, s1, s2, ... sn} where S is a Service Prototype, s is Service
Instance and D is Dynamic Proxy which is mixed of s. HI and TI means Host group id
and target group id.
W[GI] is the set of Services stored in WSDLRepository.
[:] is instantiation operator
[T] is type operator
[←] is assignment
Input : [Host Group Id] and [Target Group Id(TI), Position]
Output : Dynamic Proxies
01 foreach Sh in W[HI]
02    D[HI] ← :Sh
03    foreach St in W[TI]
04       if TSh = B then D[HI].B ∪ St
05       else D[HI].A ∪ St
06       end if
07    end foreach
08 end foreach
09 return D
```

Fig. 26. Composition algorithm in the service execution layer

Fig. 27. shows an composition algorithm in the service view layer. In the service view layer, it composes a service group that is generated by functions based on functional language. Service prototypes in the service model layer has no declarations of the functional group for the success and failure. However, in service view layer, since it uses the function as the first class object so it can dynamically create functional groups. A functional group which has the success or failure function acts as a composed service instance. Besides, it can access the configuration about the composition in runtime and can generate funcation groups.

```
Let E = {S, F}, E, F = {f0, f1, f2, ... fn} where E is a Service Execution Function
Group, S is set of succeed Functional Factor and F has opposite meaning.
W[GI] is the set of Services stored in WSDLRepository.
Input : [Host Group Id] and [Target Group Id(TI), Position]
Output : Execution Function Group
01 foreach W[HI] Sh
02    foreach W[TI] St, E
03       E.S ∪ St.S
04       E.F ∪ St.F
05       end if
06    end foreach
07 end foreach
08 return E
```

Fig. 27. Composition algorithm in the service view layer

The dynamic proxy in the service execution layer has better performance and it can be accessed as single service in the service view layer. Furthermore, it is easy to detect errors. However, it is difficult to modify byte code directly on the AOP so it is a better way to create a dynamic proxy by the Reflection. On the other hand, composition in the service view layer has lower performance and it is hard to detect errors but it provides convenience and scalability.

4) Flexible execution

Since it obtains meta data and generates WSDL by the Reflection, existing components can be published easily as web service. In this process, there is no need for additional tool and it can use existing infrastructure without modification. It supports the session to maintain context when the service has executed the overload functionality which make possible that invoke operation with the same name but different parameters in the same service instance. Moreover, it can invoke services

whether it is the local or remote service and the end-point URL for the local service is automatically generated so the service provider doesn't need to care anything about it.

5) Web service browser

Fig. 28. shows the web service browser which is fundamental service. It displays every services classified by remote and local services in the WSDL repository by the DOIMetaService of the meta service group. Since service lists are returned in form of the XML so it is transformed into user interface by using the XSLT processor service in the view layer. The web service provider selects and executes services through the template supported by framework which would be published.

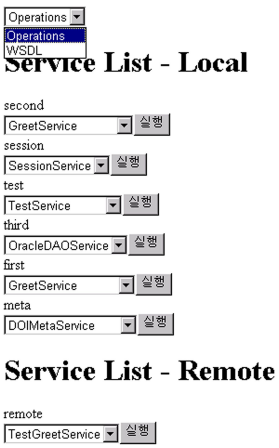

Fig. 28. Basic web service browser

Fig. 29 is an interface which enumerates operations in a service. If the operation has parameters, it supports an input form which can execute operation of the service. If the service instance is a child instance of the AbDOIService, it must have getSessionId(), isWarmed() method. It makes service instance be able to apply objects for connection to DBMS, request, response, and session.

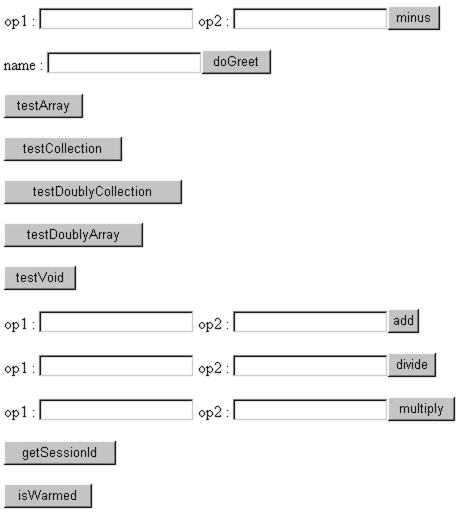

Fig. 29. Enumerate service operations

Fig. 30. shows a returned XML document from execution of the local service which directly executes a service instance.

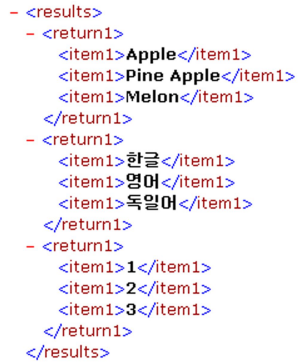

Fig. 30. Execution of the web service - local

The remote service makes equal forms with Fig. 29 but it is different when it executes an operation. Fig. 31. shows a SOAP message which returned as execution result of the remote service. A remote service extracts operations from the WSDL document and then send a SOAP request after constructing new context.

```
- <soapenv:Envelope xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:doi="http://javawide.com/DOI">
  - <soapenv:Body>
    - <ns:minusResponse xmlns:ns="http://localhost/DOI">
        <ns:return>1.0</ns:return>
      </ns:minusResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

Fig. 31. Execution of the web service - remote

Fig. 32 shows that a web service proxy is generated by referencing the WSDL document in heterogeneous .NET framework. Fig. 33. shows the returned result from the DOKDO-WS framework which has been implemented by the Java by a SOAP message.

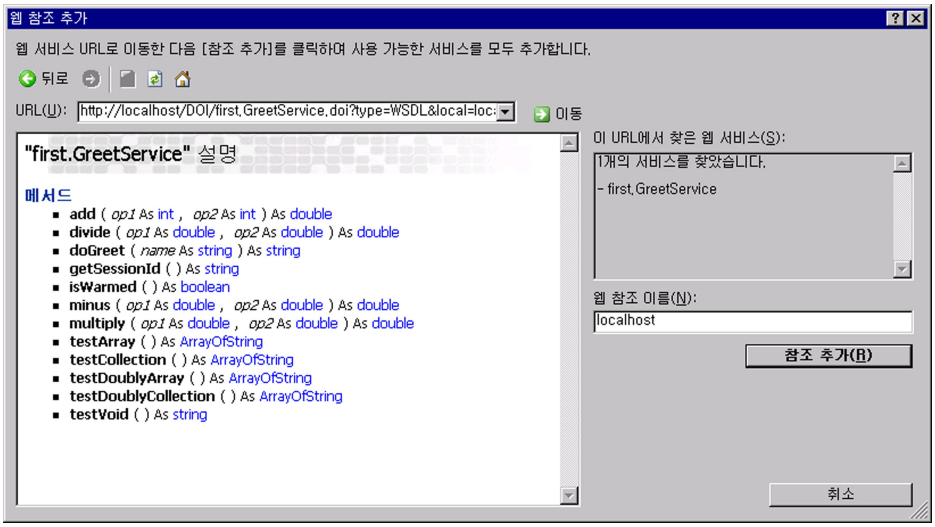

Fig. 32. Add reference to .NET framework

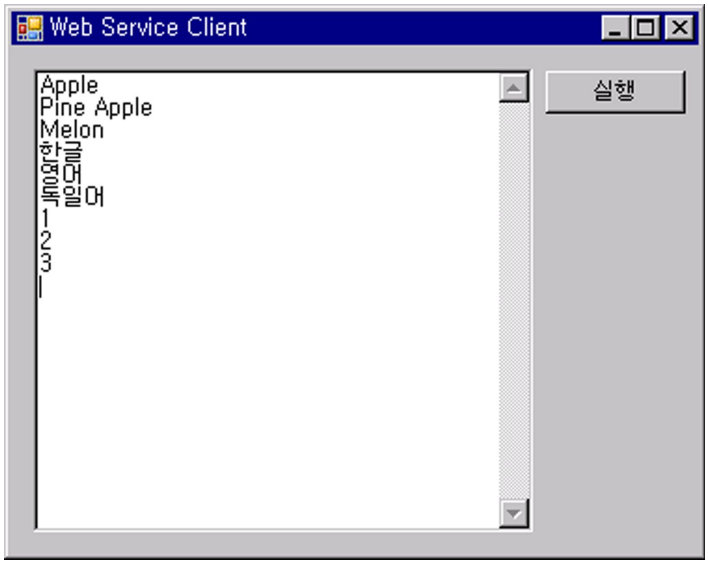

Fig. 33. Execution of the .NET web service

V. Conclusion and future work

This paper proposes the DOKDO-WS which can be applied to the web service environment without much modification on the existing method for the application development. The web service provider needs a convenient and scalable framework. However, existing web service framework hasn't satisfied with both side of effectiveness and reality and it isn't also a lightweight approach. The framework provides functionality to search and execute services and it should allow user to notify modification of the service at once in runtime. The service consumers don't want to be broken off technology between the legacy web environment and newer web service and they hope to maintain the previous HTTP request and response. Therefore, the DOKDO-WS framework supports both the SOAP and existing HTTP request, response and session to maintain state through execution of the web service.

The DOKDO-WS framework automates data modeling and persistence operations such as the INSERT, READ, UPDATE, DELETE. The business logic in the web service based application can be greatly different with what programming language is used and where it deployed but the data model is scarcely changed. The ER model is used yet and it is hard to save and restore whole semantic data models in form of the XML. In the performance evaluation, the XML based approach greatly showed poor performance on those operations. In this paper, we take both advantages that one is the performance from the ER model and another is scalability from the XML based language, which we proposed XML schema based models of the service, data, and view. Although the used domain model is not a standard such as the RDF and OWL, the service provider may manage any domain model since the DOI can handle all XML based domain model.

The future works are about the support of the standard session, the cache problem and the inference engine for semantic web. Current session implementation are partially designed and support restricted realization which can be used only in some specific environment so the standard WS-Session will be implemented in the future. As showed in the performance evaluation, the framework has higher performance compared with the others. It can obtain more performance by employing cache. The inference engine will be added to dynamic composition engine which is currently implemented and it will be used to realize semantic web.

References

[1] Okkyung Choi, Jungwoo Lee, Sangyoung Han, Advanced Web Services Retrieval System using Matchmaking Algorithm. Journal of the Korea Intelligent Information Systems Society, Volume 13. No 3 pp. 1~15. 2007

[2] Martin Treiber, Schahram Dustdar, Active Web Service Registries, IEEE Internet Computing, Volume 11, Issue 5, pp. 66 - 71, 2007

[3] Jianxun Liu, Jie Liu, Lian Chao, Design and Implementation of an Extended UDDI Registration Center for Web Service Graph, IEEE International Conference on Web Services, pp. 1174 - 1175, 2007

[4] Lin Liang, Wenge Rong, Kecheng Liu, Intelligent Agents for Pragmatic Web Services, Sixth International Conference on Advanced Language Processing and Web Information Technology, pp. 530 - 536, 2007

[5] N.W. Lo, Chia-Hao Wang, Web services QoS evaluation and service selection framework - a proxy-oriented approach, TENCON 2007 - 2007 IEEE Region 10 Conference, pp. 1 - 5, 2007

[6] Eyhab Al-Masri, Qusay H. Mahmoud, WSCE: A Crawler Engine for Large-Scale Discovery of Web Services, IEEE International Conference on Web Services, pp. 1104 - 1111, 2007

[7] Jeong-Youn Yu, Kyu-Chul Lee, Ontology describing Process Information for Web Services Discovery, The Journal of Society for e-Business Studies, Volume 12. No 3, pp. 151 - 175, 2007

[8] Hyun Namgoong, Moonyoung Chung, Kyung-il Kim, HyeonSung Cho, Yunku Chung, Effective Semantic Web Services Discovery using Usability, Advanced Communication Technology The 8th International Conference Volume 3, pp. 2199 - 2203, 2006.

[9] Assia Ben Shil, Mohamed Ben Ahmed, Additional Functionalities to SOAP, WSDL and UDDI for a Better Web Services' Administration, Information and Communication Technologies, ICTTA '06. 2nd. Vol. 1, pp. 572 - 577, 2006

[10] S.M.F.D Syed Mustapha, Relation-Based Case Retrieval Approach for Web Services Selection. IEEE/WIC/ACM International Conference on Web Intelligence. pp. 644 - 648. 2006

[11] Won-Suk Lee, Jong-Hun Park, Kyu-Chul Lee. The Protocol on WS-ECA Framework, Journal of Korean Society for Internet Information, Vol. 8. No 6. pp. 55 - 73. 2007

[12] Won-Suk Lee, Dong-Min Shin, Kyu-Chul Lee. Design and Implementation of WS-ECA Framework, The Journal of the Korean Institute of Maritime Information and Communication Sciences, Volume 11. No 8. pp.1612-1618. 2007

[13] Wu Chou, Li Li, Feng Liu. Web Services Methods for Communication over IP, IEEE International Conference on Web Services, pp. 372 - 379, 2007

[14] Yanlong Zhai, Hongyi Su, Shouyi Zhan, A Reflective Framework to Improve the Adaptability of BPEL-based

Web Service Composition, IEEE International Conference on Services Computing, Vol. 1, pp. 343 - 350, 2008

[15] Freddy Lecue, Alexandre Delteil, Alain Leger, Applying Abduction in Semantic Web Service Composition, IEEE International Conference on Web Services, pp. 94 - 101, 2007

[16] Karthikeyan Ponnalagu, N.C. Narendra, Jayatheerthan Krishnamurthy, R. Ramkumar, Aspect-oriented Approach for Non-functional Adaptation of Composite Web Services, IEEE Congress on Services, pp. 284 - 291, 2007

[17] Hui Kang, Xiuli Yang, Sinmiao Yuan, Modeling and Verification of Web Services Composition based on CPN, NPC Workshops. IFIP International Conference on Network and Parallel Computing Workshops, pp. 613 - 617, 2007

[18] San-Yih Hwang, Ee-Peng Lim, Chien-Hsiang Lee, Cheng-Hung Chen, On Composing a Reliable Composite Web Service: A Study of Dynamic Web Service Selection, IEEE International Conference on Web Services, pp. 184 - 191, 2007

[19] Wen-Yau Liang, Apply Rough Set Theory into the Web Services Composition, 22nd International Conference on Advanced Information Networking and Applications, pp. 888 - 895, 2008

[20] Il-Woong Kim, Kyong-Ho Lee, A Model-Driven Approach for Converting UML Model to OWL-S Ontology, Journal of KISS:Computing Practices and Letters, Vol. 13, No 3, pp. 179 - 192. 2007

[21] Yongyan Zheng, Paul Krause. Asynchronous Semantics and Anti-patterns for Interacting Web Services, Sixth International Conference on Quality Software, pp. 74 - 84, 2006

[22] W.L Yeung, Mapping WS-CDL and BPEL into CSP for Behavioral Specification and Verification of Web Services, 4th European Conference on Web Services, pp. 297 - 305, 2006

[23] Juil Kim, Woojin Lee, Kiwon Chong, A Tool to Construct a Web Service by Synthesizing Several Web Services, International Conference on Hybrid Information Technology, Vol 2, pp. 530 - 535, 2006

[24] Feng Liu, Gesan Wang, Li Li, Wu Chou, Web Service for Distributed Communication Systems, IEEE International Conference on Service Operations and Logistics, and Informatics, pp. 1030 - 1035, 2006

[25] Wu Chou, Li Li, Feng Liu. Web Services for Service-Oriented Communication, International Conference on Collaborative Computing: Networking, Applications and Worksharing, pp. 1 - 8, 2006

[26] Ricardo Lemos Vianna, Maria Janilce Bosquiroli Almeida, Liane Margarida Rockenbach Tarouco, Lisandro Zambenedetti Granville, Investigating Web Services Composition Applied to Network Management, International Conference on Web Services, pp. 531 - 540, 2006

[27] Sam Chung, Jennifer R. Pan, Sergio Davalos, A Special Web Service Mechanism : Asynchronous .NET Web Services, Telecommunications, International Conference on Internet and Web Applications and Services/Advanced. pp. 212 - 212. 2006

[28] Costas Vassilakis, George Lepouras, Akrivi Katifori. Web Service Execution Streamlining, International Conference on Service Systems and Service Management, Vol. 2, pp. 1564-1569. 2006

[29] Federica Paci, Mourad Ouzzani, Massimo Mecella. Verification of Access Control Requirements in Web Services Choreography, IEEE International Conference on Services Computing, Vol. 1.pp. 5 - 12, 2008.

[30] Hyun Sik Hwang, Hyuk Jin Ko, Kyu Il Kim, Ung Mo Kim. Agent-Based Delegation Model for the Secure Web Service in Ubiquitous Computing Environments, International Conference on Hybrid Information Technology, Vol. 1. pp. 51 - 57, 2006

[31] Pat. P.W. Chan, Michael R. Lyu. Dynamic Web Service Composition: A New Approach in Building Reliable Web Service, 22nd International Conference on Advanced Information Networking and Applications, pp. 20 - 25, 2008

[32] Lei Deng, Jian Wu, Zhengguo Hu. ISCF: A Semantic Web Service Composition Framework Based on OAA, The 3rd International Conference on Grid and Pervasive Computing Workshops, pp. 250 - 255, 2008

[33] Ning Gu, Juntao Cui, Wei Ye, Haixun Wang, Jian Pei. A System Framework for Web Service Semantic and Automatic Orchestration, 2nd International Conference on Pervasive Computing and Applications, pp. 606 - 611, 2007

[34] Daniel J. Mandell and Sheila A. McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation, Proceedings of the Second International Semantic Conference, Vol 2870 of LNCS. pp. 227 - 247. 2003.

[35] Rohit Aggarwal, Kunal Verma, John Miller, William Milnor. Dynamic Web Service Composition in METEOR-S, LSDIS Lab Technical Report. 2004

[36] Evren Sirin, Bijan Parsia, James Hendler. Template-based Composition of Semantic Web Services. AAAI Fall Symposium on Agents and the Semantic Web. 2005

[37] John Domingue, Liliana Cabral, Farshad Hakimpour, Denilson Sell, Enrico Motta. IRS-III: A Platform and Infrastructure for Creating WSMO-based Semantic Web Services, Proceedings of the Workshop on WSMO Implementations. 2004

[38] Denilson Sell, Farshad Hakimpour, John Domingue, Enrico Motta, Roberto C.S. Pacheco. Interactive Composition of WSMO-based Semantic Web Services in IRS-III. Proceedings of the First AKT Workshop on Semantic Web Services. 2004

[39] Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, Dana Nau. HTN Planning for Web Service Composition Using SHOP2. Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 1, No. 4. pp. 377-396. 2004

[40] Shankar R. Ponnekanti, Armando Fox. SWORD: A Developer Toolkit for Web Service Composition, Proceedings of the 11th International World Wide Web Conference, 2002

[41] Harald Meyer, Hagen Overdick, Mathias Weske. Plaengine: A System for Automated Service Composition and Process Enactment, Proceedings of the WWW Service Composition with Semantic Web Services, p. 3-12. 2005

[42] Muhammad Ahtishame Aslam, Jun Shen, Sőren Auer, Michael Herrmann. An Integration Life Cycle for Semantic Web Services Composition. 11th International Conference on Computer Supported Cooperative Work in Design, pp. 490 - 495, 2007

[43] Jin-han Kim, Chang-ho Lee, Jae-Jeong Lee, Byung-Jeong Lee. A Framework For Web Service Evolution using UML and OWL-S. Journal of Digital Contents Society, Vol. 8. No 3. pp.269-277. 2007

[44] Matt Bone, Peter F. Nabicht, Konstantin Läufer and George K. Thiruvathukal. Taming XML: Objects First, Then Markup, IEEE International Conference on Electro/Information Technology, pp. 488 - 493, 2008

[45] Konstantin Läufer. A Stroll through Domain-Driven Development with Naked Objects, Computing in Science & Engineering Vol 10. Issue 3. pp. 76 - 83, 2008

[46] Angela Nicoara, Gustavo Alonso. Making Applications Persistent at Run-time, IEEE 23rd International Conference on Data Engineering, pp. 1368 - 1372, 2007

[47] Fabio Dias Fagundez, Ricardo Niederberger Cabral, Gustavo Melim do Carmo. Decoupling Relational Database from Object Oriented Layers, Fourth International Conference on Information Technology, pp. 870 - 871. 2007

[48] Vit Vrba, Lubomir Cvrk, Vit Novotny, Karol Molnar. Architecture of a universal relation data source for web applications with advanced access control and simplified migration, Proceedings of the International Conference on Software Engineering Advances. pp. 68 - 68. 2006

[49] Adam Dukovich, Jimmy Hua, Jong Seo Lee, Michael Huffman, Alex Dekhtyar. JOXM: Java Object - XML Mapping, 8th International Conference on Web Engineering. pp. 332 - 335, 2008

[50] Andrea D'Ambrogio. A Model-driven WSDL Extension for Describing the QoS of Web Services. International Conference on Web Services, pp. 789 - 796, 2006

[51] William R. Cook, Janel Barfield. Web Services versus Distributed Objects : A Case Study of Performance and Interface Design, International Conference on Web Services. pp. 419 - 426, 2006