International Journal of u- and e- Service, Science and Technology
Vol.10, No.6 (2017), pp.71-80
http://dx.doi.org/10.14257/ijunesst.2017.10.6.07

A Novel Approach to Persistence supported Service Model based on DOI in Web Service Environments

Se-Hoon Jung[1], Eun-Cheon Lim[2], Jong-Ho Kim[3] and Chun-Bo Sim[4]

[1,3,4]*Dept. Of Multimedia Engineering, Sunchon National University, Korea*
[2] *Harvard Medical School, USA*
[1]*iam1710@hanmail.net,* [2]*euncheon_lim@hms.harvard.edu,*
[3]*jhkim@sunchon.ac.kr,* [4]*cbsim@sunchon.ac.kr*

Abstract

In design time of an application, data are belonging to specific domain and a domain object handles data in object-oriented environment. The domain object can be represented as a form of the XML in web service environment and data may be stored in the persistence layer. We propose a novel approach to persistence supported service model based on DOI (Domain Object Interface) in web serviced environments. Proposed scheme uses the Domain Object Interface to store, modify and restore domain objects. Furthermore, to manage various domains model the XML schema is applied to realize service model. Proposed service model can automate operations of the persistence layer and can handle every XML-based domain model and it shows high performance.

Keywords*: Web service, Domain object, Service model*

1. Introduction

In web services environment, the persistence layers can be in each single domain and those which have the same schema can be positioned separately in order to get higher performance and scalability. An application for web services is made up with domain data model and service model. Domain service model becomes a domain service object as same as the data model becomes data object. Both combined service objects and data objects are called domain objects. Standard web services domain model is done through the XML [1-3]. In XML-based standard and de facto standard language, ontology modeling languages such as RDF, OWL and service modeling language such as WS-BPEL, WS-CDL does not take into account the persistence layer. Standard modeling languages do only declare representation of domain objects in running time but lacks aspects for persistency. The same domain object can be stored in a distributed storage as long as the storage supports redefinition, regeneration and reinsertion with the same schema. To eliminate these repetitions, a domain model which functionality supports easy deployment and use is required. Suggested persistence supported service model in this paper provides APIs for handling XML and persistence layer. With the help of API, it generates declared domain model and creates simple links between the service objects and data objects automatically.

2. Background

Single web services are implemented under an isolated domain[4]. Therefore, data modeling within the domain while producing web services is required. Most applications alter conceptual data model to the physical schema in the DBMS[5]. Furthermore, SQL statements depending on schema and programming codes for them are written. However, such repetitive tasks tend to yield lower output and poor performance. Web services framework should reduce recurrences and offer

ISSN: 2005-4246 IJUNESST

separation of object and DBMS. For example, iBatis, Hibernate has been struggled to decouple their relation for years. ORM framework requires common model to generate representation in specific context and to create View and SQL statements based on the model in DBMS, which is mainly designed to reduce repetition. Complex operation is sometimes required depending on the model since these operations in particular model are not part of the framework and unpredictable. Aim of frameworks is to reduce foreseeable operations: insertion, modification and removal operation, and search using the keys users type. In general, data retrieval in web application occurs at the group of column marked as keys; therefore, to implement this function is painless but incremental repetition is surely painful. To solve this problem, the representation of state in applications for web services could be replaced with XML instead of the mapping between objects and XML. Definition of the class in descriptors for OR mapping is simply done by replications from the Metadata of class and Data Transfer Objects are initialized to convey the data from place to place. Most of all ideal form is XML because it is the basic unit of the message in web services environment. In [5], 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 the 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 the 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. The NaturalXML method attaches Metadata 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 a specific layer when communicating among layers. Every domain object would not keep an XML form, if an XML form is maintained by force that causes problem on performance. However, the domain object used in a web service framework should maintain the 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 an object into an XML form. In [4], 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 [5], an idea to separate service and data code is suggested.

3. Persistence Supported Service Model Based on DOI

3.1. Overall Model Structure

Figure 1 shows the service model and data model layer.

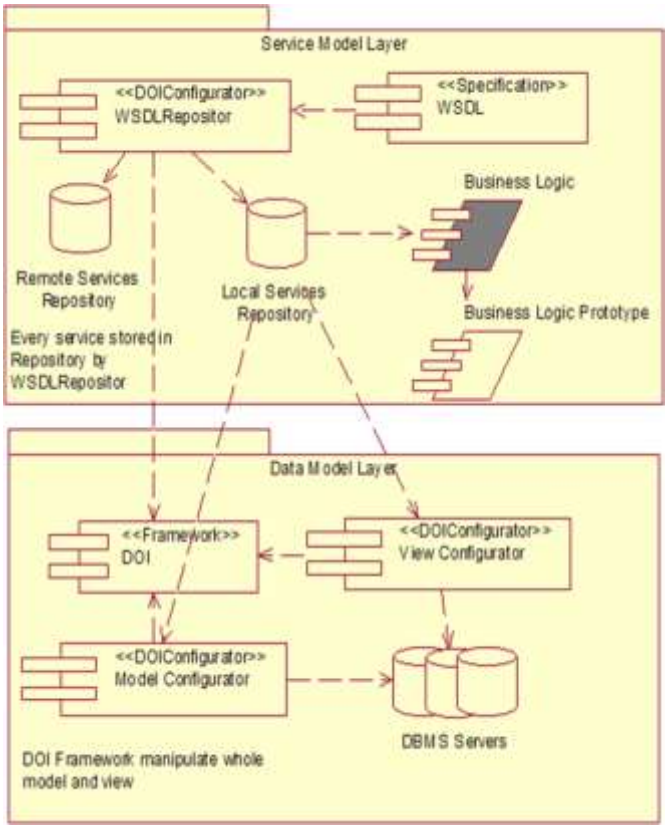

Figure 1. Service and Data Model Layer

3.1.1 Service 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 generates domain model of the business logic.

3.1.2 Data 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 when the framework initializes. Because the frameworks only register verified service to the runtime repository, service instances surely guarantees the execution.

3.2. Design of the Meta Model

3.2.1. Main Meta Model

Configurations about every domain object used in the web services are originated from the main Meta model of Figure 2. 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 Figure 2. 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.

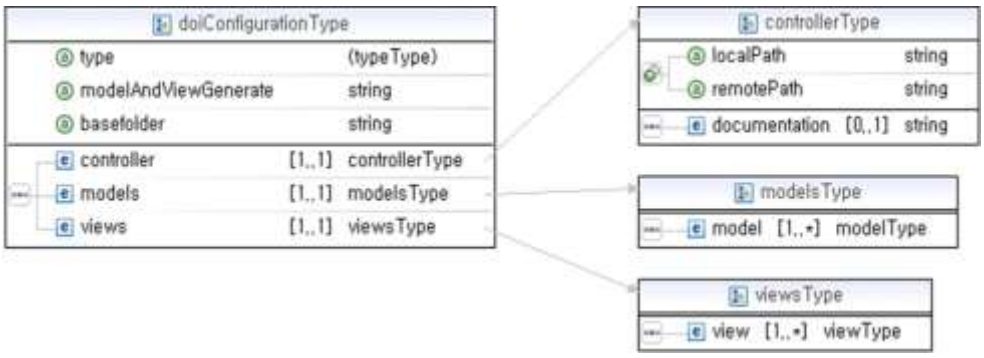

Figure 2. The Main Meta Model

Figure 3 shows model for the domain data model and view model. The domain data model and view model haves similar declaration. The domain data model expresses all information of the DBMS based persistence which is used in domain of the web service. *dbmsName* attribute can apply only the name of the DBMS which this framework supports. *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*.

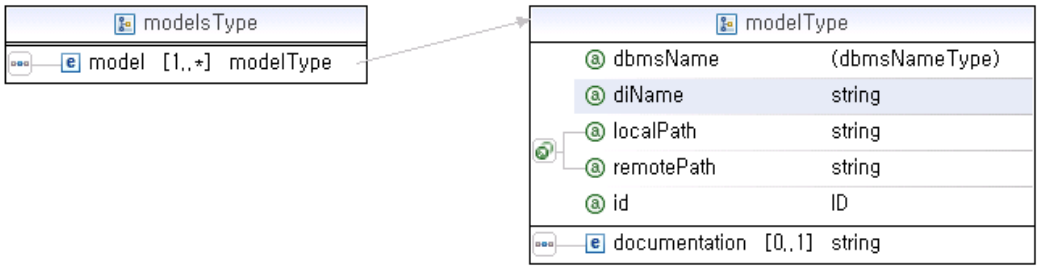

Figure 3. The Domain Data Model and View Model

3.2.2. Sub Concrete Model

Figure 4 shows the business logic model. The business logic model is a model to represent a single process which can be thought as the Controller of the MVC pattern. All business logic has equal form without any association whether it is a local or a remote service. Commonly used nodes are as follows. id attribute of controllers element, which performs as namespace and the name attribute of the controller element, which is a unique identifier of services. The documentation element describes a service and the *targetNamespace* element means a namespace of the caller. *serviceClass* element is used only in the local service and it means a module name which is employed to create a service instance. And *endPointURL* means the end-point URL to receive SOAP request from the outside. Lastly, let's look into elements which are used only in a remote service, *wsdlURL* is a URL for service description to execute.

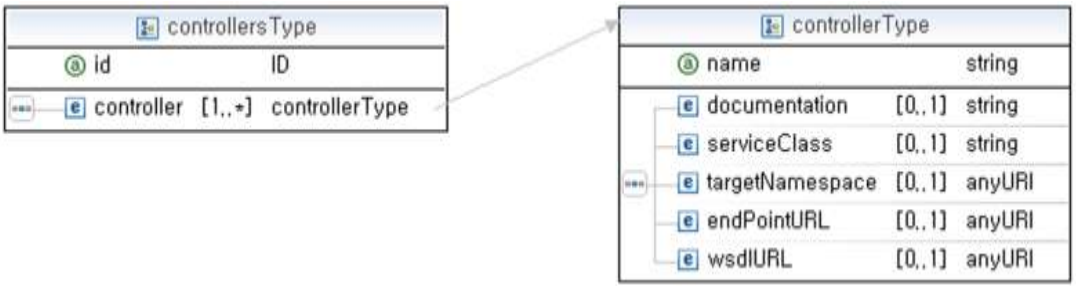

Figure 4. The Business Logic Model

Figure 5 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. Most applications basically store data in the RDBMS, thus 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. *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 makese 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.

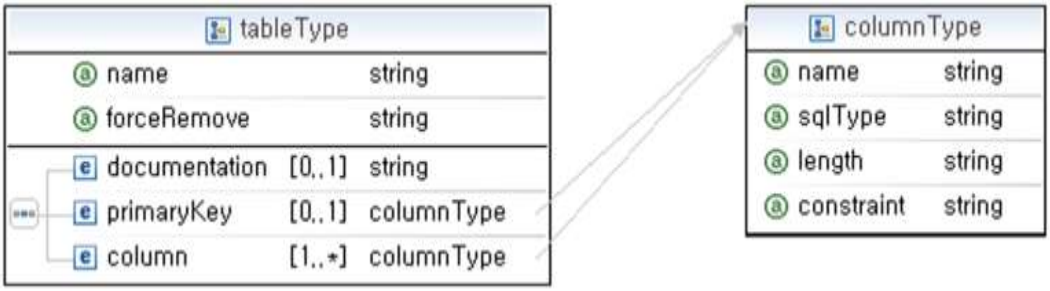

Figure 5. The Concrete Data Model

Figure 6 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 as the data model is done. *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 other columns to generate operations.

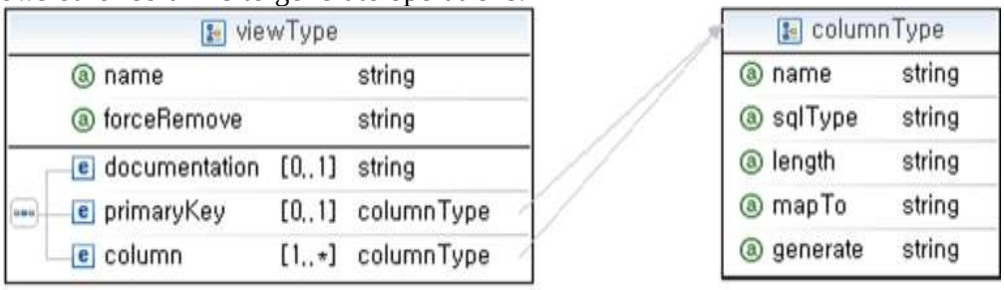

Figure 6. The Concrete View Model

4. Implementation and Performance Evaluation

In the performance evaluation of this paper, we use the Intel Core i5 64 bit 3.6Ghz for the CPU and the Windows7 Professional SP1 for the OS and the JDK 1.8.0 version for the programming environment. We use the HttpClient 4.0 Beta 1 version to request specific service. 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.

4.1. Model Implementation

Figure 7 is an example of a data model instance. Database elements contain several table elements that contain column element. Column element defines columns of a single table and constraint attribute specifies a constraint on that column.

```
<doim:doi xmlns:doim="http://javawide.com/DOImodel" type="model" xmlns:xsi
="http://www.w3.org/2004/XMLSchema-instance" xsi:schemaLocation="http://ja
vawide.com/DOImodel http://javawide.com/DOImodel">
<doim:database name="Bank">
<doim:table name="Account" forceRemove="true">
<doim:documentation>Account for Bank Service</doim:documentation>
<doim:primaryKey name="account_number" sqlType="BIGINT"/>
<doim:column name="balance" sqlType="Numeric(19,2)" constraint="DEFAULT 0
CHECK(0 &lt;= balance)" />
</doim:table>
<doim:table name="Account History" forceRemove="true">
<doim:column name="account_number" sqlType="BIGINT" constraint="FOREIGN
KEY REFERENCES Account (account_number) ON DELETE CASCADE"/>
...
</doim:table>
<doim:table name="Customer" forceRemove="true">
<doim:documentation>Customer for Bank Service</doim:documentation>
...
<doim:column name="social_security" length="14" sqlType="CHAR"
constraint="CHECK(social_security LIKE '[0-9][0-9][0-9][0-9][0-9][0-9]-[0-9][0-9][0-9][0-
9][0-9][0-9][0-9]')"/>
</doim:database>
```

Figure 7. An Instance of Data Model

4.2. Performance Evaluation

Figure 8 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.

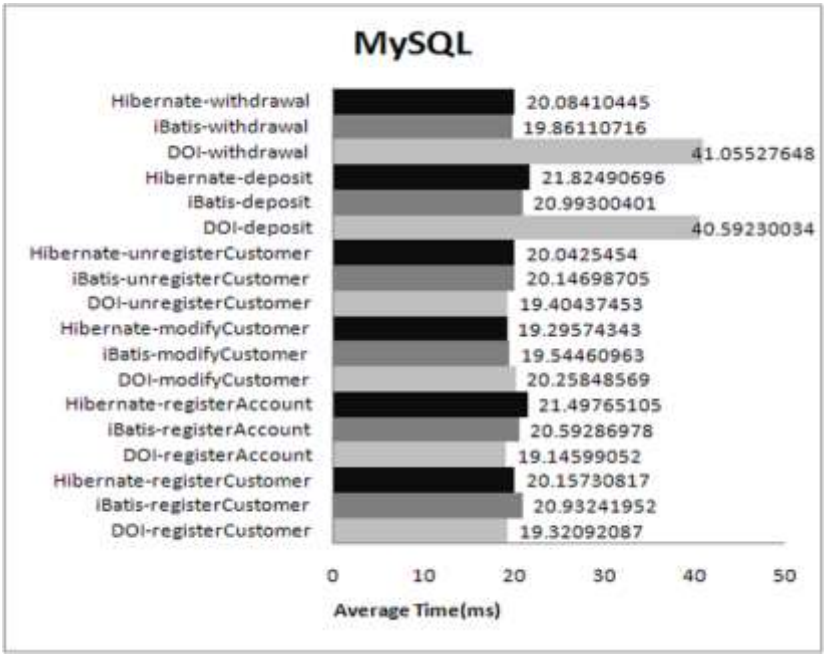

Figure 8. INSERT, UPDATE, DELETE Operations in the MySQL

Figure 9 shows performance evaluation of the INSERT, UPDATE, DELETE, and COMPOSED operations in the MS-SQL DBMS and the JOXM. *registerCustomer* operation inserts a row of the 9 columns and *registerAccount* operation inserts a row of the 5 columns. JOXM is influenced enormously by the number of columns. *modifyCustomer* which is the UPDATE operation is 4 times lower in the JOXM and 6 times in the others than INSERT operation. *unregisterCustomer* which is DELETE operation is 4 times lower than INSERT operation. 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 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.

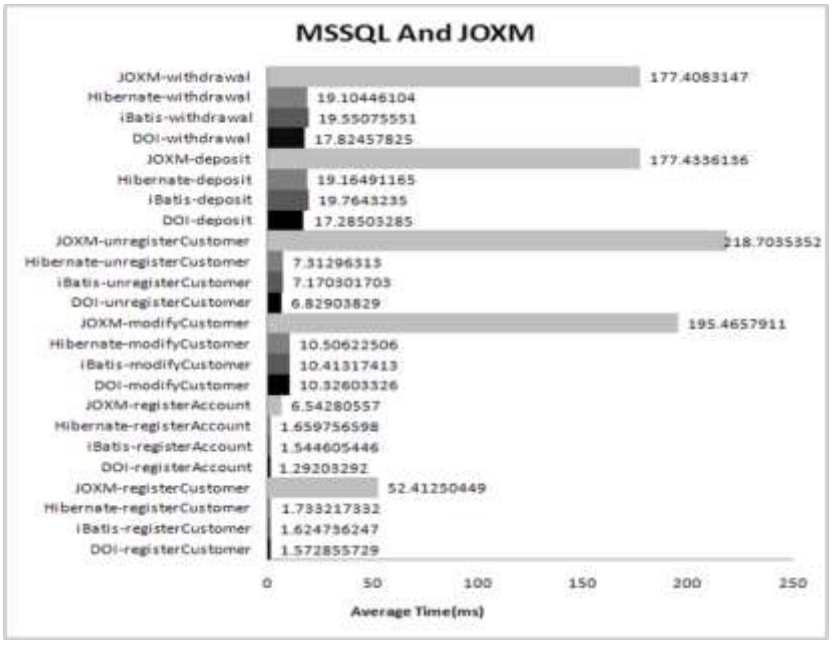

Figure 9. INSERT, UPDATE, DELETE operations in the MSSQL and JOXM

Figure 10 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.

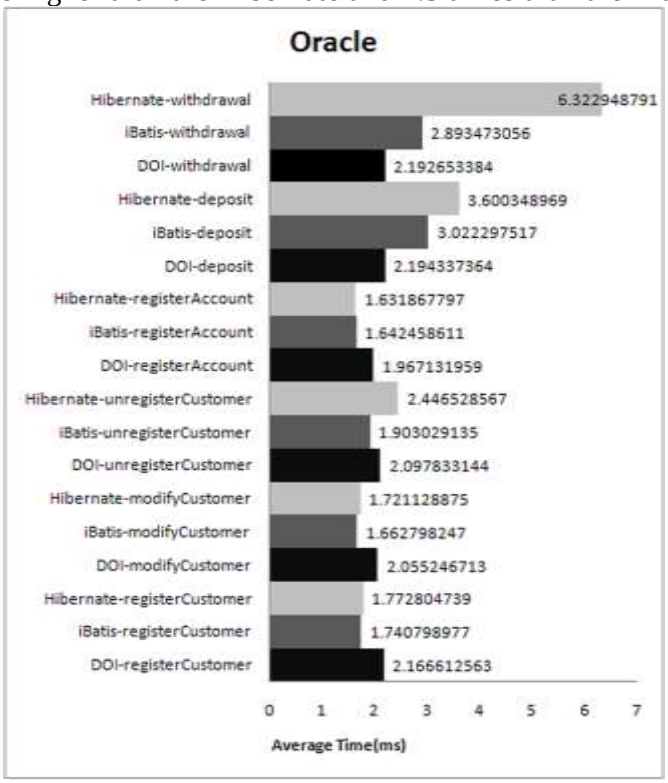

Figure 10. INSERT, UPDATE, DELETE Operations in the Oracle

5. Conclusion and Future Work

In this paper, a service model that automates the domain objects to be stored, modified, restored in a variety of business processes is proposed. Users of the service model can generate operations used by persistence layer with a little XML declaration. In addition, since every XML language is manageable, it could provide compatibility with the standard domain modeling language. In order to evaluate the performance of the service model proposed, the comparison is performed in terms of INSERT, UPDATE, and DELETE operations with other ORMs such as iBatis, Hibernate. This model yields high performance with Oracle, MSSQL, MySQL. In future research, we are going to perform optimization for the persistence operations by implementing appropriate DBMS in the distributed environment.

References

[1] M. Bazarganigilani, B. Fridey and A. Syed, "Web Service Intrusion Detection Using XML Similarity Classification and WSDL Description", International Journal of u- and e- Service, Science and Technology(IJUNESST), vol. 4, no. 3, **(2011)**, pp. 61-72.
[2] M. K. Yusof, A. F. A. Abidin, S. M. Deris and S. Usop, "Implementing of XML and Intelligent Algorithm for Improving Web Query Processing in Heterogeneous Database Access", International Journal of Database Theory and Application(IJDTA), vol. 4, no. 2, **(2011)**, pp. 13-22.
[3] S. Shahriar and J. Liu, "Towards the Preservation of Referential Constraints in XML Data Transformation for Integration", International Journal of Database Theory and Application(IJDTA), vol. 4, no. 2, **(2010)**, pp. 1-10.
[4] K. Läufer, "A Stroll through Domain-Driven Development with Naked Objects", Computing in Science & Engineering, vol.10, **(2008)**, pp. 76-83.
[5] V. Vrba, L. Cvrk, V. Novotny and K. 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, Papeete, Tahiti, **(2006)**.

Authors

Se-Hoon Jung, received his BSc and MSc in Multimedia Engineering from Sunchon National University in 2010 and 2012, respectively. Currently, he is a team manager with the research & development team, Gwangyang Bay SW Convergence Institute, South Korea. His research interests include data analysis and data prediction. E-mail : iam1710@hanmail.net.

Eun-Cheon Lim, received his BEng. and MEng. in Computer Science and Multimedia Engineering from Sunchon National University, South Korea, in 2007, and 2009, respectively. He received his PhD in bioinformatics from Max Planck Institute and University of Tuebingen, Germany, in 2016. He is now a research fellow at Harvard Medical School, USA. His research interests lie in molecular biology, bioinformatics algorithms. E-mail : euncheon_lim@hms.harvard.edu

Jong-Ho Kim, received a BSc, MSc, and PhD in electronic communication engineering from Hanyang University, South Korea, in 1998, 2000, and 2008, respectively. Currently, he is an associate professor with the Department of Multimedia Engineering, Sunchon National University, South Korea. His research interests include image/video compression, and processing. E-mail: jhkim@sunchon.ac.kr

Chun-Bo Sim, received a BSc, MSc, and PhD in Computer Engineering from Chonbuk National University, South Korea, in 1996, 1998, and 2003, respectively. Currently, he is an associate professor with the Department of Multimedia Engineering, at Sunchon National University, South Korea. His research interests include multimedia databases, ubiquitous computing systems, and big data processing. E-mail: cbsim@sunchon.ac.kr