Applying Model-Driven Approach to Building Rapid Distributed Data Services

Chih-Min LO¹, Member and Sun-Jen HUANG¹†, Nonmember

SUMMARY The globalization of commerce has increased the importance of retrieving and updating complex and distributed information efficiently. Web services currently show that the most promise for building distributed application systems and model-driven architecture is a new approach to developing such applications. The expanding scale and complexity of enterprise information systems (EISs) under distributed computing environments has made sharing and exchanging data particularly challenging. Data services are applications tailored specifically for information oriented tasks to deal with business service requirements, and are heavily dependent on the distributed architecture of consumer data processing. The implementation of a data service can eliminate inconsistency among various application systems in the exchange of data. This paper proposes a data-oriented model-driven developmental framework to deal with these issues, in which a platform independent model (PIM) is divided into a service model, a logic data model, and a service composition model. We also divide a platform specific model (PSM) into a physical data model and a data service model. In this development method, we define five metamodels and outline a set of rules governing the transformation from PIMs into PSMs. A code generator is also included to transform each PSM into the application code. We include a case study to demonstrate the feasibility and merits of the proposed development framework with a case study.

key words: data services, web services, model-driven development, unified modeling language, model transformation

1. Introduction

Enterprise information systems (EISs) are used widely to process massive quantities of data in the support of corporate operations. The need for closer collaboration within corporations and pressure due to globalization has made EIS increasingly complicated, and therefore, the demand for data integration is greater than ever before. EISs are fundamentally based on data processing. Over the past 20 years, various information systems have been developed to manage data and information from different perspectives [25]. Processing data from different sources requires the integration of various forms of hardware, operating systems, applications, and databases, which involves the transformation, mapping, aggregation, and transfer of data. Nevertheless, most applications closely associate data access with business logic. Data processing logic becomes redundant when multiple applications are integrated, because each application development team implements its own data processing logic, which resulting in problems related to the exchange and integrity of data.

Data Services is a convenient mechanism to provide a service interface to access data from a relational database. Accordingly, providing data as a service not only encourages the access to data anywhere at any time but also reduces the cost of an information system implementation [22]. Data services are tailored for information oriented tasks to deal with data requirements for business operations, relying on the distributed architecture of consumer data processing mechanisms. The implementation of a data service can resolve inconsistencies between various applications involved in the exchange of data.

Service Oriented Architecture (SOA) is a new paradigm for distributed computing and electronic commerce, enabling the establishment of a distributed collaboration network to integrate various discrete functions within an enterprise or organization. Data services serve to differentiate data integration from business applications, which enhances the flexibility of processing and managing information. Web Services, based on SOA distributed applications, receive particular attention recently as a viable means to develop data services [23]. The model-driven approach is a proven software development methodology proposed by the Object Management Group (OMG) [16] to improve the portability of software.

Figure 1 presents the architecture of a typical data service application. A data service provides an interface for any typical application system such as rich client, Web client, and text model application; moreover, it only involves in the constructing of data access layer comprising data process logic and does not concern with the implementations of application logic and user interfaces. Therefore, constructing of the data services should include two implementations: database and data service application. In the past, many researchers proposed methodologies for the construction of databases and a few of the methods dealt with the issue of constructing data retrieve models and model transformation [2], [9], [21], [24]. A number of researches made many attempts in the area of applying the model-driven approach to construct SOA-based applications. However, most of which have focused on the business services development [7], [8] and did not consider transforming data models into working code of the data services application.

To overcome these problems, this study applies the model-driven approach to the development of a quick and efficient mechanism for developing distributed data service applications. We propose a data-oriented framework,
comprising modeling tools, methods, and code generators. We employ a unified modeling language (UML) [18] extension mechanism to define a set of meta-models for design purposes, and outline transformation rules using the ATLAS Transformation Language (ATL) [4], [12] to enable the automatic transformation of models. The main contribution of this study is the development of a model constructing specification and transformation rules that overcome issues associated with the rapid building of the database and data services application.

The remainder of this study is structured as follows. Section 2 provides background to the current research. Section 3 discusses the literature related to methodologies for the development of model-driven applications and web services. Section 4 presents an overview of the proposed framework for building distributed data services rapidly. Section 5 illustrates meta-models for constructing models. The model transformation rules and procedure are described in Sect. 6. The final section provides a discussion for demonstrating the value of the proposed framework and presents our conclusions.

2. Background

Service-oriented computing (SOC) is an emerging distributed computing architecture for e-business, enabling the construction of distributed applications within and across organizational boundaries for purposes of collaboration. Web services are currently the most promising technologies based on SOC, operating as self-describing, self-contained, modular components capable of being published, located, and invoked across the web using standard interfaces and protocols [25]. Web services are web-based applications comprising coarse-grained business functions accessed through the internet [6]. Today, web service technology is important for building distributed data service applications in the integration of enterprise applications.

The model-driven architecture (MDA) initiated by the OMG is a conceptual framework for software development that promotes the efficiency of developing software and ensures the quality and maintainability of the resulting product. This approach requires effective techniques for implementing code generators for domain-specific languages [10]. MDA covers three basic models: CIM, PIM, and PSM. Code generators are used to transform PSMs into implementation code for specific platforms, and the data exchange protocol XMI (XMI, XML Metadata Interchange) promotes the exchange of data between modeling tools and code generators.

UML is a general purpose language used for visual modeling. Despite being the most widely used modeling language in software development; it does not satisfy all information modeling needs. Thus, UML 2.0 was developed to provide two kinds of extensions: a heavyweight extension method based on the direct modification of the UML meta-model; and a lightweight extension that allows system analyzers to adapt the semantics of UML without having to alter the UML meta-model [18].

Model transformation technology is a key factor in MDA. The success of the Query/Views/Transformations (QVT) [19] initiative has made it a possible avenue for model-driven development. The final adopted specifications for QVT are currently an OMG standard for defining model transformations. This QVT specification has a hybrid declarative and imperative nature resulting from the three QVT languages. These languages are Relations, Core, and Operational Mappings. Relations and Core are declarative languages at two levels of abstraction. Operational Mappings is an imperative language extending from the Relations and Core languages.

The ATLAS transformation language (ATL) is a hybrid language used to model transformations [12]. ATL was inspired by the OMG QVT requirements and constructs based on the formalism of object constraint languages (OCLs) [17]. ATL was initially conceived as an answer to QVT RFP but later the language requirements evolved towards a larger set of transformational scenarios. ATL provides two imperative constructs: the rule and action block. The rule makes explicit requests similar to procedures, but its body may comprise declarative target patterns. The action block is a sequence of imperative instructions that can be used in either matched or called rules. Transformation programs written in ATL are inherently unidirectional [13].

3. Related Work

Many recent studies have appeared in the fields of service-oriented computing and model-driven development [5], [11], [14], [20], with most of this research focusing on the construction of web service description language (WSDL), rather than the implementation of web services applications. Unfortunately few of these studies have dealt with the integration of data services with model-driven development (MDD) [1], [15]. This study refers to web service implementation techniques, database modeling, and approaches to model transformation.

Zhang (2008) stated that current research can be divided into model driven technologies for SOA and data-centric applications. Modeling SOA applications involves the use of UML and business process model (BPM) with business process execution language (BPEL). Database modeling uses conceptual, logical, or physical data models. To satisfy the requirements of data services, these two areas
must be connected to improve the traceability and agility involved in the movement of data. Zhang (2008) proposed a unique information liquidity model to deal with the different aspects of data integration to improve on traditional data modeling technology [26].

Castro et al. (2011) proposed a method for the service-oriented development of information systems. Their method follows an MDA-based approach to provide a complete set of CIM and PIM meta-models as well as the specifications of the mappings between them. They demonstrate how a model-driven approach can be used to deal with the problem of alignment between the business view and the information system view that arises when adopting a service-oriented approach in software development [4]. Nevertheless, their work was not related to the field of database modeling integration. Model transformation is a key to realizing MDA. Jouault et al. (2008) presented a model transformation language and execution environment based on the Eclipse framework. They demonstrated how ATL tools could support major tasks involved in using a language: editing, compiling, executing, and debugging [12].

Databases contain two schemas: Table is a physical schema for storing data; View is a virtual table without an actual schema or data assembled by a database query operation. Today, most software developers use UML class diagrams to model database models. Ambler (2003) [2] and Gornik (2003) proposed UML data modeling profiles for relational database modeling [9]. These two modeling methods are similar in that they use UML extension mechanisms to define a UML profile for relational database modeling. The respective UML profiles are used to define a set of stereotyped classes such as Table and View to describe database schemas. Class attributes are used to denote a simple column for the database Table or View. Computed columns are represented by OCL expressions, and dependency relationships are used to illustrate the relationship of derivation. Unfortunately, these two methods have not reached the stage of transformations from model to model. This study also defines a set of transformation rules for transformations from logic data models to physical data models.

4. Overview of the Framework

This study proposes the framework in Fig. 2 for the model driven development of data services. This framework divides the process of system development into two stages. The PIM model construction stage defines two models from different perspectives: the service model and logic data model. For the service model, we utilize the construction language in the UML use case diagram. This model is used to define the service required by the business from a functional perspective. The logic data model lacks details related to database manipulation and is used to define the data structure required to support the business service. We employ the model construction language in the UML class diagram for this model.

PSM is the second stage of model construction in the MDD and includes three models: a physical data model, a service composition model and a data service model. The physical data model describes the database schema in the implementation of the database. The service composition model is designed from the perspective of data manipulation, utilizing UML sequence diagram to describe the data processing from the aspect of service. The data service model defines the details of system implementation, and we utilize UML class diagram for model construction in this model. To clearly define the modeling in this framework, we utilized the UML extension mechanism to define five meta-models. These meta-model provide guidelines for the construction of the model, as well as the mapping element for conversion from model to model. We also define a set of model transformation rules for the automation of model transformation to support the implementation of our framework. PSM in the proposed framework can be processed using the proposed code generator to obtain working code.

This paper focuses on the model driven development of data centric information systems to implement rapid development. The modeling methodology is centralized around information. During system development, PIM modeling is performed to rapidly identify the consumer within the system and the business service to be constructed. Analysis of the information requirements is then performed to construct a data model based on the service defined in the first step. A logic data model (LDM) is developed independently of the database implementation. In the modeling stage, an original LDM was constructed by manual process to meet certain business requirements and subsequently transformed this original model into one or more models. Once the LDM is complete, we will transform LDMs into a physical data model (PDM) and several service composition models (SCMs) according to the model transformation rules. After SCM construction is done, the final step is to
convert these models to the data service model (DSM) in accordance with the transformation rules we defined. DSM includes two packages: a data service package and a data contract package. Following completion of the final stage of modeling, the code generator is used to generate code for the data services program and the SQL code for specific platforms.

5. Meta-Models for Model Constructions

A UML meta-model provides a specification for model constructing, and containing many meta-classes and relationships. A meta-class states a stereotype, −has attributes and is represented by a rectangle notation. An attribute of the meta-class is illustrated by a string which comprises a name and a data type. In the modeling, a meta-class shows a stereotype; an attribute of a meta-model is a tagged-value. In a UML model, a stereotype is represented as a string between a pair of guillemets (« ») or as a new icon. A tagged value specifies a new kind of property that it is attached to a model element and rendered as a string enclosed by a pair of braces ({}). A constraint is a specification for model elements, attached to any model element to refine its semantics, and can be defined by means of an informal explanation using Natural Language or by means of OCL expressions. It is also rendered as a string enclosed by a pair of braces ({}).

Figure 3 shows the relationship between a meta-class and a class.

5.1 Meta-Model of Service Models

SM is the first system built using the proposed framework. It is implemented from a functional perspective with the aim of defining the business service to be developed. Figure 4 illustrates the meta-model in this SM, including the two elements, Consumer and Service. This meta-model is an extension of Kernel and UseCase meta-models in the UML standard. The Consumer is inherited from Actor, to represent the user of a system function. A Service is inherited from UseCase to describe a business function. The DirectedAssociation is used to describe the relationship between Consumer and Service. A Consumer can be related to multiple Service elements.

Figure 5 shows an example of the service model which conforms to the meta-model of service models shown in the Fig. 4. This example model indicates a service scenario that a buyer places an order. BookApp is an actor of the consumer illustrating a subscriber of the data service, and it is also a remote information system. This element in a service model is notated by a UML actor notation with a «Consumer» stereotype. BookService states a data service for indicating a data service provider. In the construction of a service model, a data service provider modeled by using a UML uses case notation with a «Service» stereotype. A directed association states a relationship of the subscription.

5.2 Meta-Model of Logic Data Models

LDM describes the data required for processing in the system and the relationships within the data. Figure 6 shows an LDM meta-model, which is an extension of Kernel package in the UML standard. An Entity meta-class is defined in the
meta-model. The Entity element inherits from Class, and is composed of at least one attribute. An Entity can have zero or multiple keys. The Entity can relate to relationships including: Association, Aggregation, Composition, Generalization, and Dependency. The Dependency only connects from a derived Entity class to another Entity class, and is divided into two kinds of Dependency relationships: from and joined.

Figure 7 shows an example of the LDM. This model conforms to the meta-model of LDM shown in the Fig. 6. It contains two example classes and one relationship for presenting. A publisher should publish one or more book. Book is an «Entity» class modeled by using Entity meta-class in the LDM meta-model. A Book class comprises three attributes according to the Attribute meta-class of the LDM meta-model. A key attribute of the «Entity» class is represented by using a string with a key tagged-value (\{key\}). An association relationship linked between Book and Publisher conforms to Association meta-class of LDM meta-model.

5.3 Meta-Model of Physical Data Models

PDM is a data model transformed from LDM which describes the composition relationship of the database schema. PDM has two class elements: Table and View. Two different data schemas are shown in Fig. 8. A Table is composed of at least one Column, and can have zero or multiple keys. The relationship between two Table elements can only be one of the two DirectedAssociation relationships: Cascade or NoAction. These two associations are used to describe the constraint relationship between database tables. View is used to describe the database schema of a virtual table, which is also compositied by Columns. View can only have one of the two Dependency relationships: Derived or Joined. Derived is inherited from the Dependency relationship, which has two end points: source and target. The source end can only be connected to a View element, and the target end can be connected to Table or View.

An example shown in the Fig. 9 presents a PDM construction conforms to the PDM meta-model. Book and Publisher are «Table» classes modeled by using Table meta-class of the PDM meta-model. A «NoAction» directed-association between Book and Publisher is modeled by using NoAction meta-class. BookView is a «View» class conformed to the View meta-class. The two dependency relationships conform to the Derived and Joined meta-class. Attributes of the Book conforms to Column meta-class.

5.4 Meta-Model of Service Composition Models

Figure 10 shows a SCM meta-model, it describes the relationship between a service and the data required to implement the service. This model is also transformed from LDM. SCM has two elements: Service and Entity; and two message elements: Update and Retrieve. It requires at least two object elements one Service and one or more Entity object. In the SCM, the message only connects the Service object to Entity object. An Update message is an interaction between a Service and a not derived Entity object for representing a data manipulation. A Retrieve message shows the data retrieving and is only connected to a derived Entity object.

Figure 11 shows an example of the SCM, conforming to the SCM meta-model shown in the Fig. 10. BookService...
is «Service» object conforming to Service meta-class of the SCM meta-model. Book and BookView are the «Entity» objects that are modeled by the Entity meta-class. The «Update» message line is constructed by the Update meta-class, and «Retrieve» message line is constructed by the Retrieve meta-class.

5.5 Meta-Model of Data Service Models

DSM is a PSM that is transformed from a SCM describing the details of the system implementation. DSM includes a WebService element for the implementation of a data service. A WebService is connected to at least one DataAccess object. A WebService contains one or more WebMethod operation. A WebMethod is a data operation in the data service model that implement for a remote system call to trigger a data manipulation. Figure 12 shows the meta-model of DSM. A DataAccess class can instance as a data access object in the data access layer of the application, and it is related at least one DataEntity class which is similar to a data structure mapping to a database schema. A DataEntity class also comprises a set of methods for performing data manipulations from a data persistency.

A DSM construction conforms to the DSM meta-model. The example model of the DSM shown in Fig. 13, presents a combination of data service comprising four classes. BookService is a «WebService» class constructed by the WebService meta-class. BookDataAccess states a class modeled by using the DataAccess meta-class. Book and BookView are constructions modeled by using the DataEntity meta-class. Two «Use» directed-association are implemented from the Use meta-class; and one directed-association is modeled by using the DirectedAssociation meta-class.

5.6 Model Verification Rules

In this section, we present the verification rules declared by OCL to verify models using our method. Once the model has been designed, we use several criteria to verify it to ensure the accuracy of model transformation. Analyzing OCL invariants can reveal insightful information and ensure the correctness of the model transformation [3]. Therefore, this study declares a number of OCL invariants to specify verification rules for checking database models. We describe the verification rules and the implementation of OCL invariants. Due to limitations in the number of pages, we are unable to list all of the OCL expression. The summary of the logic of the LDM verification rules are listed below:

1. An instance of the Entity class includes at least one attribute, and has two tags comprising Derived and Primary. These two kinds of tags is of Boolean value (Line 2–5).

2. An attribute of the instance of the Entity class has one tag with a Boolean value, and the tag is named “key” (Line 6).

3. The instance of the Entity class without a Derived tagged-value can only be related to the instances of the relationships which comprise Association, Aggregation, Composition, Generalization (Line 7–10).

4. An instance of the Entity class with a Derived tagged-value can only be related to the instance of the Dependency relationship (Line 11–13).

5. A multiplicity of the relationship is a type of enumeration and the value must be one of the three kinds comprising one-to-one, one-to-many, and many-to-one (Line 14–16).

The following OCL expressions are some of logic of the LDM verification rules:

1. Context Entity inv:
   2. self.attributes->size() > 0 and
   3. self.attributes->forAll(c | c.oclIsKindOf(Attribute)) and
   4. self.Derived.oclIsKindOf(Boolean) and
   5. self.Primary.oclIsKindOf(Boolean) and
   6. self.keys->forAll(k | k.oclIsKindOf(Attribute) and k.key = true) and
   7. if self.Derived <> true then implies
   8. self.related->forAll(r | r.oclIsKindOf(Association) or r.oclIsKindOf(Aggregation) or
   9. r.oclIsKindOf(Composition) or r.oclIsKindOf(Generalization))
   10. endif
   11. if self.Derived = true then implies
   12. self.related->forAll(r | r.oclIsKindOf(from) or
6. Model Transformation

The most important process in model-driven software development is model transformation. Model transformation refers to the transformation of a source model to another target model using a set of transformation rules and mapping mechanisms. With regard to the definition of mapping, Miller and Mukerji [16] stated that, “the mapping description may be in natural language, an algorithm in an action language, or a model in a mapping language”. According to MDA, a model-driven procedure is a model transformation between various levels of abstraction. Accordingly, we define meta-models at various levels of abstraction, taking into consideration the procedure of transformation between models. For the model transformation, we use natural language and ATL language to describe the process of transformation and define the model transformation rules. Figure 14 describes a model transformation pattern based on ATL. In this pattern, the source model is transformed into the target model according to the ATL transformation rules. The model and transformation rules originate from its meta-model, and each meta-model conforms to MOF standards. According to ATL approach, the transformation defined in ATL form modules. A module contains a mandatory header section, import section, and a number of helpers and transformation rules [12].

In order to improve the clarity of the presentation, we brief a case study in below and this case is a part of a real case of an e-commerce information system. Figure 15 illustrates an original LDM class diagram for presenting a partial data entities of the bookstore information system which constructed by manual and conforms to our defined meta-model shown in the Fig. 6. This model comprises six original entities (Book, Publisher, Author, Order, Buyer, and Person) and two composite entities (Item and BookAuthor) which connected to the source end of a Composition or an Aggregation relationship.

6.1 Transformation from LDM into LDMs

An original LDM is the first construction of data model in the PIM modeling stage, and this model only contains entities for representing data persistency elements. After the constructing, we automatically transform the original data model into one or more data model with a tool support for indicating each elementary data access. Once this transformation is completed, we add derived entities by manually to meet the requirements of the data retrieving. For this transformation, we summarize transforming procedure and rules by natural language. The transformation rules and procedure are the below:

Step 1: Create a new model for each original Entity class. Then, transform this entity into a new Entity class with a Primary tagged-value.

Step 2: Transform the many-to-one Association and related entity which connected to the entity of a new model.

Step 3: Transform the Composition relationship and related entity which connected to the entity of a new model with the end of the composition.

Step 4: Transform the Aggregation relationship and related entity which connected to the entity of a new model with the end of the aggregation.

Step 5: Transform the Generalization relationship and related entity which connected to the entity of a new model with the end of the source.

Step 6: Repeat the step 2-5.

Step 7: Create a derived entity for primary entity and its composite entities which connected with a many-to-one Association or a Generalization relationship, and then create a from dependency relationship to like the derived entity and the source entity.

Step 8: Create a joined dependency relationship for each related entity which connected to a source entity of the derived entity.

Figures 16–21 show models which transformed from an original LDM shown in the Fig. 15 and this transformation follows Step 1 mentioned above. Figure 16 shows a Book model transformed from the Book entity and follows Step 2–8. Publisher and Author are the entity which trans-
6.2 Transformation from LDM into PDM

This process involves the automatic conversion of LDM (at a higher level of abstraction) to PDM (at the level of physical database implementation). To this end, we define the transformation procedure and mapping rules, the meta-model of which is shown in Figs. 6 and 8. There is only one Entity class in the LDM and two classes Table and View in PDM. Each Entity can be transformed into a Table or a View according to the Derived tagged-value. There are two Dependency relationships in PDM for the View element: Derived and Joined. A Derived relationship links from a View to a Table transformed by a from Dependency relationship. A Joined relationship links to another Table from a View that it is transformed by the joined Dependency relationship. The following are the transformation rules and procedure.

Step 1: Transform each not derived Entity into a Table.
Step 2: Transform each derived Entity into a View.
Step 3: Transform each attribute into a Column, and converts a key attribute to a PK column.
Step 4: Transform each relationship into a NoAction directed association except the Composition relationships, and directed to the Table which transformed from an Entity connected to the end of one side of many-to-one multiplicity.
Step 5: Transform each Composition relationship into a Cascade directed association, and directed to the Table which transformed from an Entity connected to the end of composition side.
Step 6: Transform each from Dependency relationship into a Derived relationship.
Step 7: Transform each joined Dependency relationship into a Joined relationship.
Step 8: Append a PK column of the Table to the associated Table classes and set key value equal FK with a tagged-value.

Figure 22 shows a PDM which is transformed from
Fig. 15. In this mode, all of the Table and View classes are converted by Step 1–2. The NoAction associations are transformed from relationships by Step 4, and the Cascade associations are transformed by Step 5. The dependency relationships are transformed by Step 6–7. Step 8 transforms the FK column from a PK column.

6.3 Transformation from LDM into SCM

SCM is a model transformed from an LDM, and describes the communication relationship between service and data entity. This model conforms to a meta-model as Fig. 6 shown. In this section, we utilize natural language to summarize transformation rules for illustrating the logic of transformation. To present a clear explanation, a case study is provided to show the details of the transformation procedure. The following are the summarization of the model transformation rules.

Step 1: Create a Service object for the primary Entity class.
Step 2: Transform each entity into the Entity objects.
Step 3: Create Update message lines for connecting the Service object and each not derived Entity objects.
Step 4: Create Retrieve message lines for linking the Service object and each derived Entity objects.

Figure 23 shows a Book SCM which is transformed from an LDM shown in Fig. 16. In this case, the model contains a Service object, four Entity objects. This model comprises two Update message lines and two Retrieve message lines. BookService is a Service object for illustrating a data access object which transformed by Step 1 mentioned above. Book, BookView, BookAuthor and BookAuthorView are objects which transformed from Book LDM model, and this transformation is following Step 2. The message line 1 and line 3 are the Update message lines which are created by using transformation Step 3. Message line 2 and 4 are the Retrieve message lines, and these two message lines are created according to the transformation Step 4.

6.4 Transformation from SCM into DSM

DSM is a model for implementing data service applications, and conforms to a meta-model which has shown in Fig. 12. A DSM is transformed from a SCM and contains three types of classes includingWebService, DataAccess, and DataEntity classes. A WebService class is a boundary of data service application for providing functions to the remote system call, and is transformed from a Service class of the SCM. The DataAccess class is a controller and also transformed from a Service class. All DataEntity classes are transformed from the Entity class and List class. The transformation rules and procedure are the below:

Step 1: Transform the not derived Entity class into a DataEntity class including all attributes, then add five data manipulating methods to the class.
Step 2: Transform the derived Entity class into a DataEntity class including all attributes, then add two methods for retrieving data.
Step 3: Transform the Service class into a DataAccess class, then create the Use directed association relationships and connect it to each DataEntity class.
Step 4: Add attributes to the DataAccess class that each attribute is similar to a variable declaration of a DataEntity class. Then, add methods to the class for indicating the data manipulations.
Step 5: Transform the Service class into a WebService class; add an attribute and methods to the class. The attribute is a type of DataAccess class.
Step 6: Create a directed association, and connect from the WebService class to a DataAccess class.

Figure 24 shows a Book DSM transformed from a SCM which is shown in Fig. 23. The DataEntity classes Book and BookAuthor transformed by Step 1. Both classes BookView and BookAuthorView are which transformed from two derived classes and by Step 2. The class BookDataAccess transformed from the BookService class by Step 3 and 4. The BookService is a WebService class which also transformed from a BookService class in the SCM as Fig. 23 shown.
6.5 Transformation from PDM into SQL

SQL statements are divided into two major categories including data definition language (DDL) and data manipulation language (DML). A DDL is for constructing database schema purpose, and DML is used to retrieve data from a database. In this transformation, we focus on the generation that to produce the statements of DDL. Figure 25 shows a template of SQL statement for creating a table schema. Most of the tables should add a primary key constraint, as shown in Fig. 26, which is a template of SQL statement for constructing a primary key constraint to a table. Figure 27 is a template of the SQL data definition statement for adding a foreign key constraint to a table. Figure 28 states a template of SQL statement for constructing a view schema.

A summarized transformation process shows the brief main transforming logic and rules. This brief description listed on the below.

Step 1: First, to transform each Table class from a PDM into a SQL DDL code.

Step 2: Then to transform PK columns of a Table class into a SQL statement for adding a primary key constraint to its own table.

Step 3: If a table contains FK columns then to transform these FK columns into a SQL statement for adding a foreign key constraint into this table. A foreign key constraint may with the referential actions, which address by the tagged-values that owned by a reference which connected to this table with a source end.

Step 4: Final step is to transform each View class from the PDM into a SQL create view statement.

The details of transformation rules are given by the ATL program code, and in this section we will list a part of programs from all transformation program codes. The following code only presents the logic of the parts of the transformation program and transformation rules. Line 3–22 is a transformation program for transforming a Table class into a SQL DDL statement, and these lines of code refer to a data type converting function that is also defined in the same module as Line 25–39 shown. Line 40–51 presents a SQL generation that produces a SQL statement for adding a primary key constraint into a table with PK columns, and this procedure refers to a primary key list generating function which defines in Line 52–63. However, in this study we focus on the description of the logic of the transformation rules, and therefore we only show a part of all transformation programs. Figure 29 shows partly codes of SQL statement which are transformed from the PDM shown in Fig. 16 by using the transformation rules mentioned above.

```
CREATE VIEW <view name> ( 
    <column name 1>,
    ...
    <column name n>
) AS
SELECT <fromTable.columnName 1>, ..., 
    <fromTable.columnName 2>, ..., 
    <joinTable.columnName 1>, ... 
FROM <from table name>
    INNER JOIN <join table name> on <join condition>;
```

**Fig. 28** A Template of SQL create view statement.
Fig. 29 SQL DDL statement.

```java
22. let filePath = path + t.name + '.sql'
23. Sql.writeToFile( filePath );
24. );
25. helper context def : ConvertToSqlDataType( srcType : String ) : String =
26. let tarType : String = new String(
27. switch srcType {
28. case 'String' then
29. tarType = 'VARCHAR(250)'
30. case 'Integer' then
31. tarType = 'Int'
32. case 'DataTime' then
33. tarType = 'DateTime'
34. case 'Boolean' then
35. tarType = 'bit'
36. default
37. tarType = 'VARCHAR(250)'
38. }
39. return tarType;
40. helper context def : CreatePK( ) =
41. let allTables : PDM!Table = PDM::Class.allInstances( )->collect()
42. foreach t : PDM!Table in allTables |
43. let sql : String = new String( )
44. sql += 'ALTER TABLE ' + t.name
45. sql += ' ADD CONSTRAINT PK_' + t.name
46. sql += 'PRIMARY KEY ('
47. sql += GetPKs( t )
48. sql += ');'
49. let filePath = path + 'PK_' + t.name + '.sql'
50. Sql.writeToFile( filePath );
51. );
52. helper context def : GetPKs( t : PDM!Table ) : String =
53. let cols : PDM!Column = t.columns->select( c | c.key = 'PK')
54. let pk : String = new String( )
55. let count : Integer = 0
56. foreach c : PDM!Column in cols |
57. pk += c.name
58. count++
59. if count < cols->size() then
60. pk += ', '
61. end
62. )
63. return pk;
```

6.6 Transformation from DSM into Code

The application code is a final production in the software development. In our proposed framework, the application program code is generated from the DSMs constructed by using a meta-model, as shown in Fig. 12. In order to facilitate a clear description of the application code transformation, we use the same case mentioned above. In this case, we separate the application code into three packages: web service package, data access package, and data entity package.

A package of the application is similar to a component library that provides a set of services for a specific domain. This study builds the constructions by using CSharp language and Microsoft .NET framework. Figure 30 shows an application code which is transformed from the Book DataEntity class shown in Fig. 18. The transformation rules and procedure are shown as below:

**Step 1:** The first step is to create a folder, and creating a solution file in the folder. Then create three sub-folders in this solution folder, and three project files in each sub-folder. These three folders are namely DataEntities, DataAccessed, and DataServices. These three project files are also named by using the folder’s name with a “.csproj” file extension.

![Fig. 30 The application code — DataEntity Class.](image-url)
Step 2: The second step is the first transformation that to transform each DataEntity class into a data entity class file, and declares variables of a class according to the columns of the class. Then, add functions into a class file according to the methods of a class.

Step 3: Transform each DataAccess class into a data access class file is the third step, and the process of this transformation is the same as the second step.

Step 4: Finally, transform each WebService class into a single class file and save to the DataServices folder; the process of transformation same as the second step.

As to the following code we also adopt ATL helper declaration as an algorithm for presenting logic of the model transformation. In this section, we provided an example to introduce the generating of a data entity class file, and we also utilize the example mentioned in Fig.24 to offer a clearer description. Line 3–46 is the main procedure to create a data entity class file, and Line 52–63 is to generate a parameter list of a function. Line 64–86 shows a procedure for generating a data manipulation SQL statement.

```csharp
// Module declaration
module DSM2CSharp;
// Create OUT: CSharp from IN: DSM;

// Helper context def: CreateDataEntityFile( )
let allDataEntities : DSM!DataEntity = DSM!Class.allInstances()->collect();
for de in allDataEntities |
  let code = String = new String();
  code.append('using System;') |
  code.append('using System.Data;') |
  code.append('namespace DataEntities {') |
  foreach c in de.columns |
    code.append('public ' + c.datatype + ',' + c.name + ';') |
  code.append('} namespaces DataEntities { ');
  code.append('} (DataContract)'); |
  code.append('public class ' + de.name + 'DataEntity {') |
  foreach m in de.methods |
    code.append('public ' + m.datatype + ',' + m.name + ';') |
  code.append('} DataMember{DataMember} = new DataMember();') |
  if m.name = 'add' |
    code.append(GetSqlAddStatement( de ); + ') + ' + m.name + ');') |
  code.append('int result = db.executeNonQuery( sql, this );') |
  code.append('return result;'); |
  else if m.name = 'update' |
    code.append(GetSqlUpdateStatement( de ); + ') + ' + m.name + ');') |
  code.append('int result = db.executeNonQuery( sql, this );') |
  code.append('return result;'); |
  else if m.name = 'delete' |
    code.append(GetSqlDeleteStatement( de ); + ') + ' + m.name + ');') |
  code.append('int result = db.executeNonQuery( sql, this );') |
  code.append('return result;'); |
  else if m.name = 'retrieve' |
    code.append(GetSqlRetrieveStatement( de ); + ') + ' + m.name + ');') |
  code.append('DataSet ds = db.executeQuery( sql, this );') |
  code.append('return ds;'); |
  else if m.name = 'list' |
    code.append(GetSqlQueryStatement( de ); + ') + ' + m.name + ');') |
  code.append('DataSet ds = db.executeQuery( sql, this );') |
  code.append('return ds;'); |
  code.append('}'); |

// Call to the generated class file
let fileName : String = de.name + '.cs';
file.write(fileName);
```

7. Discussion

The limitations of our research are: (a) Data of the software development efforts is hard for us to collect, because we are unable to find out the related researches. To show the power of the modeling and transformation, we provide three projects we executed in Taiwan for human effort evaluation in the development stage (see Table 1); (b) the proposed transformation method is a unidirectional transforming approach.

Table 1 shows the lines of code of the three projects we experienced. The Project A contains 157 files and 19115 lines of code (LOC), in which 15770 LOCs are generated from the models with our proposed method and 3345 LOCs by manual. Project B includes 146 files and 12491 lines of code (LOC), in which 10400 LOCs are generated from the

| Table 1 | The development effort of three real projects. |
|-------------|----------------|
| Project | File count | Total LOC | Generated LOC | User code | Generated code rate | User code rate |
| A | 157 | 19115 | 15770 | 3345 | 82.50% | 17.50% |
| B | 146 | 12491 | 10400 | 2091 | 83.26% | 16.74% |
| C | 85 | 6206 | 5168 | 1038 | 83.27% | 16.73% |
| Average | | | | | 82.88% | 17.12% |

Note:
1. LOC: Line of code
2. The generated code rate shows the percentage of the reduced development effort.
3. The formulas: d=(b/a)*100, c=(c/a)*100
models with our proposed method and 2091 LOCs by developers’ effort. Project C has 85 files and 6206 line of code (LOC), in which 5168 LOCs are generated from the models with our proposed method and 1038 LOCs by human effort.

According to the result of the LOC analysis, we found that our proposed method to develop the data services results in reducing human effort by 82.88% in the coding stage. On the contrary, the development effort exerted without the proposed method is 100%.

The projects provided are: Project A is an e-commerce system of a bank which we have finished in Taiwan; Project B is an e-commerce system working in EVAN Educational Institute in Taiwan (www.evan.com.tw); Project C is an e-commerce system working in Taiwan Law Journal Co., Ltd. (www.taiwanlaw.com.tw).

8. Conclusion and Future Work

Data services serve as a new form of application for the rapid implementation of distributed information systems that facilitate sharing and integration of data between applications. The model driven approach is a fast and effective means of building a data service application, relying on model construction and transformation techniques, as well as code generators. Despite recent progress in this area, integrating database modeling and model transformation to develop data services remains a challenge. This study proposes an MDA-based developmental framework used to construct distributed data services applications, with an emphasis on the mapping between PIMs and PSMs. A case study was adopted to illustrate how the proposed system deals with the implementation of mapping rules between two levels of abstraction.

We recognize that it is the key for the rapid software development to develop a mechanism which is capable of reducing the efforts, raising the quality, and advancing the software maintainability. The basic processes for rapid constructing a data access application includes constructions such as logic data model, basic data retrieval model, physical data model, application class model, SQL code, and application code. Table 1 describes a result of the development effort with and without the proposed method. The comparison illustrates that to develop the data services with our proposed method can reduce effort by 82.88% in the coding stage. On the contrary, the development effort without the proposed method should be 100%. Accordingly, we can justify that this method is to EIS developers constructing distributed data services. The advantages of adopting our proposed method to construct data services are the below:

1. Reduce the efforts of development: The software developers only construct the data models, and then transform increasingly these models into SQL code and programming code with a tool support. At this point, the developers can save programming time and reduce the efforts.

2. Raise the quality of software: SQL and application code is generated by a model transformation process, and this way does not occur to the defects. On the contrary, the coding by manual process it is possible to occur to the defects.

3. Advance the software maintainability: The developer only needs to maintain a higher level of abstraction data model, without the need to maintain the code. Accordingly, we believe that the software maintainability is higher than traditional development methods.

The case study we present only exemplifies the implement in the platform of the conventional Web application with a web browser. Confined by the limitation of time and space of the study, in the present study, there is no case study for the other two platforms, a rich client platform and text-based UI for batch processing. Our model transformation method is a unidirectional transforming. We will solve these limitations in our anticipating work.

References

[1] P.V. Andukuri, J. Guo, and R. Pamula, “Automatic Integration of web services,” Proc. 2011 International Conference on Internet Technology and Applications (iTAP), pp.1–4, 2011.

[2] S.W. Ambler, Agile Database Techniques, Wiley Publishing, Canada, 2003.

[3] J. Cabot, R. Clarisó, E. Guerra, and J. de Lara, “Verification and validation of declarative model-to-model transformations through invariants,” Journal of Systems and Software 83, pp.283–302, 2010.

[4] V.D. Castro, E. Marcos, and J.M. Vara, “Applying CIM-to-PIM model transformations for the service-oriented development of information systems,” Information and Software Technology, vol.53, no.1, pp.87–105, 2011.

[5] Y. Chen, B. Wu, and B. Xia, “Design of web service single sign-on based on ticket and assertion,” Proc. 2011 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), pp.297–300, 2011.

[6] J.Y. Chung, K.J. Lin, and R.G. Mathieu, “Web services computing: Advancing software interoperability,” Computer, Oct. 2003, pp.35–37, 2003.

[7] A. Fuhr, T. Horn, et al., “Model-driven software migration into service-oriented architectures,” Computer Science - Research and Development, pp.1–20, 2011.

[8] R. Gronro, D. Skogan, et al., “Model-driven Web services development,” Proc. 2004 IEEE International conference on e-Technology, e-Commerce and e-Service (EEE’04), pp.1–4, 2004.

[9] D. Gornik, UML Data Modeling Profile, TP162, 05/02, IBM Rational Software, Available from: http://www3.software.ibm.com/ibmdl/pubs/softwarerational/web/whitepapers/2003/tp162.pdf, 2003. (Accessed 11 July 2010)

[10] Z. Hemel, L.C.L. Kats, D.M. Groenewegen, and E. Visser, “Code generation by model transformation: a case study in transformation modularity,” Software and Systems Modeling, vol.9, no.3, pp.375–402, 2010.

[11] Y. Jiao, L. Li, and N. Ye, “Towards a Lightweight SOA Framework for Enterprise Cloud Computing,” Proc. 2011 15th International Conference on Computer Supported Cooperative Work in Design, pp.386–392, 2011.

[12] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, “ATL: A model transformation tool,” Science of Computer Programming 72 (2008), pp.31–39, 2008.

[13] F. Jouault and I. Kurtev, “On the interoperability of model-to-model transformation languages,” Science of Computer Programming, vol.68, no.3, pp.114–137, 2007.
[14] N. Loutas, V. Peristeras, and K. Tarabanis, “Towards a reference service model for the web of services,” Data & Knowledge Engineering, vol.70, no.9, pp.753–774, 2011.

[15] C. Mayr, U. Zdun, and S. Dustdar, “View-based model-driven architecture for enhancing maintainability of data access services,” Data and Knowledge Engineering, vol.70, no.9, pp.794–819, 2011.

[16] OMG, Model-Driven Architecture Guide Version 1.0.1, Object Management Group, Inc., Available from: http://www.omg.org/cgi-bin/doc?omg/03-06-01, 2003. (Accessed 11 July 2010)

[17] OMG, Object Constraint Language Version 2.0, Object Management Group, Inc., Available from: http://www.omg.org/spec/OCL/2.0, 2006. (Accessed 11 July 2010)

[18] OMG, Unified Modeling Language: Superstructure version 2.1.1, Object Management Group, Inc., Available from: http://www.omg.org/cgi-bin/doc?formal/07-02-05, 2007. (Accessed 11 July 2010)

[19] OMG, Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Available from: http://www.omg.org/spec/QVT/1.1/PDF, Jan. 2011, (accessed 2011/12/6)

[20] X. Qafmolla and N. Viet Cuong, “Automation of Web services development using model driven techniques,” Proc. 2nd International Conference on Computer and Automation Engineering (ICCAE), pp.190–194, 2010.

[21] T. Roelleke, H. Wu, et al., “Modelling retrieval models in a probabilistic relational algebra with a new operator: the relational Bayes,” The VLDB J., vol.17, no.1, pp.5–37, 2008.

[22] H.L. Truong and S. Dustdar, “On analyzing and specifying concerns for data as a service,” Proc. 2009 IEEE Asia-Pacific Services Computing Conference (APSCC 2009), Dec. 2009, pp.87–94, 2009.

[23] W.T. Tsai, R. Paul, L. Yu, A. Saimi, and Z. Cao, “Scenario-based web services testing with distributed agents,” IEICE Trans. Inf. & Syst., vol.E86-D, no.10, pp.2130–2144, Oct. 2003.

[24] J.H. Wu, Y.C. Huang, et al., “Object-oriented analysis and design: Transformation from class diagram to relational table and application template.” J. Internet Technology, vol.6, no.4, pp.453–462, 2005.

[25] X. Yu, Y. Zhang, T. Zhang, L. Wang, J. Hu, J.H. Zhao, and X. Li, “A model-driven development framework for enterprise Web services,” Information System Front 2007(9) 391–409.

[26] X.G. Zhang, “Model driven data service development.” IEEE International Conference on Networking, Sensing and Control, pp.1668–1673, 2008.

Chih-Min Lo is currently a PhD candidate in the Department of Information Management, National Taiwan University of Science and Technology, Taiwan. He received his B.A. in Department of Chinese Literature in 1990 from Soochow University, Taiwan, and Master degrees in Information Management in 1996 from National Central University, Taiwan. He is also a member of the Software Engineering and Management Laboratory (SEML), which participates in several projects from National Science Council, Taiwan. His main research interests include software engineering, information systems modeling, object-oriented technology, and database management system. He is currently an associate professor of technology staff and director in the Department of Information Management, Chungyu Institute of Technology, Taiwan.

Sun-Jen Huang received his B.A. in Industrial Management in 1988, and his M.S. in Engineering and Technology in 1991, both from the National Taiwan Institute of Technology, Taipei, and the PhD degree from the School of Computer Science and Computer Engineering, La Trobe University, Melbourne, Australia, in 1999. He is currently a professor in the Department of Information Management, National Taiwan University of Science and Technology, Taipei. He is also the head of the Software Engineering and Management Laboratory (SEML), which hosts several research projects from National Science Council, Taiwan. Dr. Huang is also a member of Software Quality Promotion Committee at the Chinese Society for Quality. His research interests include software measurement and analysis, software process improvement, software quality Assurance, and software project management. Dr. Huang has published papers in journals including IEEE Transactions on Software Engineering, IET Software, Information & Management, Software Practice & Experience, Journal of Systems and Software, Information and Software Technology, Industries Journal, and Applied Intelligence, etc.