A Template-based Model Transformation Approach for Deriving Multi-Tenant SaaS Applications

Kun Ma[1], **Bo Yang**[2], **Ajith Abraham**[3, 4]

[1] Shandong Provincial Key Laboratory of Network Based Intelligent Computing, University of Jinan, 250022 Jinan, China
e-mail: ise_mak@ujn.edu.cn

[2] Shandong Provincial Key Laboratory of Network Based Intelligent Computing, University of Jinan, 250022 Jinan, China
E-mail: yangbo@ujn.edu.cn

[3] Machine Intelligence Research Labs, Scientific Network for Innovation and Research Excellence, 98071 Auburn, USA
e-mail: ajith.abraham@ieee.org

[4] IT For Innovations, VSB - Technical University of Ostrava, Ostrava - Poruba, Czech Republic

Abstract: Software-as-a-Service (SaaS) and Model-Driven Engineering (MDE) are two of the most dominant software engineering paradigms nowadays. Multi-tenancy is the key to successful SaaS. In this paper, we introduce a data middleware to customize the multi-tenant database first. In addition, with the help of model transformation, it is possible to generate SaaS applications from the models. However, most of the current model transformation approaches do not fully support the requirements for model synchronization, and they do not cater for the specific problems faced in the multi-tenancy. Therefore, an effective and simple template-based model transformation and model synchronization approach based on model evolution of MDE paradigms is fully integrated for the development of SaaS multi-tenant applications. The proposed framework uses a novel extensible business component model (xBC) to sufficiently describe both the structural and behavioral properties of SaaS applications. The distribution and uninterrupted running of the generated SaaS applications proves that our approach is feasible and correct in practice.

Keywords: Software-as-a-Service; multi-tenancy; textual template evolution; model transformation; model synchronization

1 Introduction

Model-Driven Engineering (MDE) is becoming the dominant software engineering paradigm to specify, develop and maintain software systems, mainly because it can raise the level of abstraction and automation in software construction [1]. Some findings on experiences from using model-based development in industry from the EA-MDE project indicate that 83% of our questionnaire respondents think that MDE improved productivity and maintainability [2]. The use of MDE has the following consequences for a software development process [3]: 1) More time can be devoted to analyzing the business; 2) The time needed to perform coding tasks is reduced; 3) Productivity is improved as the time necessary for coding is reduced.

Software-as-a-Service (SaaS) is a software delivery on-demand model in which software and its associated data are hosted centrally in the cloud. According to International Data Corporation's (IDC) latest market report, SaaS will grow at a 26.4 percent compound annual growth rate (CAGR) through 2015[4]. As SaaS of the cloud infrastructures is the future tendency of the IT industry, it is urgent to research on the generation approach of SaaS applications. A recent survey of organizations with experience using cloud applications and platforms reveals that the most urgent need is how to tightly integrate it with other applications and how to convert the legacy systems into SaaS applications [5].

Therefore, it is natural that we wonder how both paradigms, MDE and SaaS, can be integrated and benefit from each other. Bruneliere et al. discuss two different collaboration scenarios between MDE and SaaS [6]: 1) MDE for the cloud refers to the use of MDE techniques to facilitate and (semi)automate the development of SaaS applications. 2) MDE in the cloud involves using cloud infrastructure to enable MDE in new and novel ways, corresponding to on-demand Modeling as a Service (MaaS) initiative. Similar to SaaS, MaaS would allow the deployment and on-demand execution of modeling and model-driven services over the Internet. In accordance with scenario 1, we aim to identify opportunities for using MDE to support the development of cloud-based SaaS multi-tenant applications. Therefore, this paper proposes a transparent SaaS multi-tenant data middleware, which is fully integrated with template-based model transformation approach and model synchronization based on model evolution of MDE paradigms for the development of SaaS multi-tenant applications.

The rest of the paper is organized as follows. Section 2 discusses the background and related work. In Section 3, an extensible business component model (*xBC*) is presented to describe SaaS multi-tenant business and database to the fullest. The architecture of multi-tenant data middleware and *xBC* is discussed in detail. In Section 4, a template-based model transformation approach that supports model synchronization is presented to generate the SaaS application. Conclusions are provided in the last Section.

2 Related Works

The major problem of SaaS modeling lies in the customization of the data and business.

2.1 Multi-tenant Data Model

An important requirement for SaaS applications is the support of multiple tenants [7]. Data architecture is an area in which the optimal degree of isolation for a SaaS application can vary significantly depending on technical and business considerations. An overview of approaches for data management in a multi-tenant deployment can be found in [8] and [9]. The paper categorizes existing approaches of shared applications and briefly explains them in Figure 1, each of which lies at a different location in the continuum between isolation and sharing. 1) **Separate schema with shared application** involves housing multiple tenants in the same database, with each tenant having its own set of tables, which are grouped into a schema created specifically for the tenant. Unfortunately, this approach tends to lead to higher costs for maintaining equipment, backing up tenant data and restoring data in the event of a failure. The number of tenants that can be housed on a given database server is limited by the number of schemas that the server can support. 2) **Shared schema with shared application** involves using the same database and the same set of tables to host multiple tenants' data. A given table can include records from multiple tenants stored in any order; a Tenant ID column associates every record with the appropriate tenant. The shared schema approach has the lowest hardware and backup costs. However, this approach may incur additional development effort in the area of security, to ensure that tenants can never access other tenants' data. Compared with the two approaches, we lead to improvements in the relational database, and propose multi-tenant data middleware.

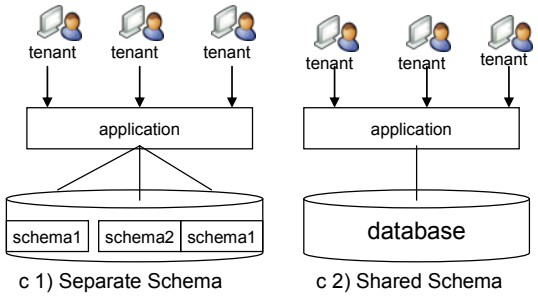

Figure 1
The common SaaS multi-tenant data models

2.2 SaaS Application Modeling

Several researchers have proposed using variability modeling techniques from software product line engineering in the context of service-based systems. Chang et al. address the problem that the variability of business processes and services is not explicitly modeled, which hinders implementing adaptive service-based systems [10]. They extend the XML schemas of service description languages in order to cater for variability. Liu et al. propose a new modeling method for constructing SaaS Service using extended Web Services Conversation Language (WSCL) [11]. Wang et al. propose a service community model based on eXtensible Markup Language (XML) for a bilateral SaaS mode which is abstracted from a real project of the nationwide service network for sharing science and technology information [12]. Although these approaches propose explicitly documenting variability, they do not cater for the specific problems faced in the SaaS context (e.g., multi-tenancy). Motivated by these problems, the extensible business component model (*xBC*), based on the extension of the Unified Modeling Language (UML) profiles, is abstracted from SaaS applications to describe the multi-tenant business to the utmost.

2.3 Model Transformation Approach in Support of Model Synchronization

Model transformations are essential in the process of MDE [13]. The development of a software system is an iterative process with frequent modifications to the involved models according to the user requirements [14]. As a consequence, an effective and simple model transformation methodology that supports model synchronization is needed urgently. However, most of the current model transformation approaches have some limits, such as fully incremental support for model synchronization. Additionally, the behavior of the SaaS application cannot be modeled in detail, in which case it is easier to write source code manually. This means that the mixture often leads to a wide range of inconsistencies [15]. Therefore, this information should be kept during the model transformation, and several possibilities exist to develop model transformations for the sake of model synchronization. An overview of model transformation and synchronization systems can be found in [16]. As outlined in the introduction, MDE requires a bidirectional solution which preserves model contents when synchronizing as much as possible. However, many available model transformation approaches only support classical one-way batch-oriented transformations [17]. This basic feature updating existing target models based on changes in the source models is also referred to as *change propagation* in the Query/View/Transformation (QVT) final adopted specification [17]. The QVT implementation [18] is only unidirectional but partly incremental. Other existing TGG-based approaches also do not provide a comparable automatic and computational incremental solution

(for a detailed discussion see [19] and [20]). Compared to these approaches, our approach of model transformation that supports model synchronization is based on model evolution. This approach of model synchronization will only take the storage space of model repositories rather than extra space. Only the models with changed version number need a subsequent model transformation. This method is named *source incrementality*, which is simple and useful for working with large scale source models. In this way, model synchronization is a special and partial model transformation. This is a good way to minimize the amount of source that needs to be reexamined by a transformation when the source is changed.

3 Extensible Business Component Model in Support of Multi-Tenancy

The important features of SaaS applications are multi-tenant data and business customization.

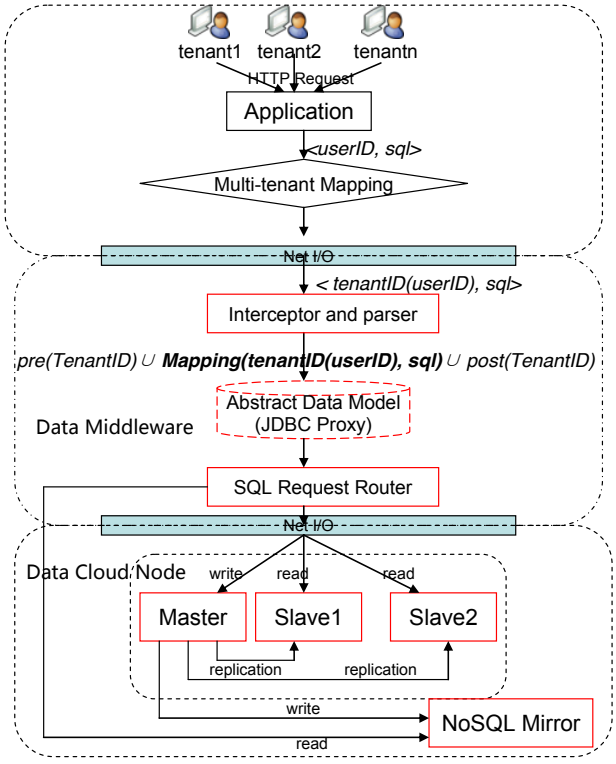

Figure 2
Multi-tenant Data Middleware

3.1 A Transparent Multi-Tenant Data Model

In this section, we propose a data middleware to customize the multi-tenant database. The architecture of data middleware shown in Figure 2 is comprised of the Abstract Data Model, the JDBC Interceptor and parser, the SQL Request Router and the Data Cloud Node.

3.1.1 Abstract Data Model

The Abstract Data Model acts as database JDBC proxy without the storage of any data for the sake of smooth transition. It is transparent to the application, owning the same set of tables and views as the physical database. Therefore, the application can connect to the abstract data model without any modification. All the application lifecycle management procedures (upgrade or patch) may remain as they are. The abstract data model provides the logical data isolation for the tenants with higher demand on security.

3.1.2 The JDBC Interceptor and Parser

The JDBC Interceptor and parser is used to intercept the SQL and formulate the new SQL to the Abstract Data Model. The new SQL is transformed from the original SQL and tenant information. The interceptor process is denoted as $sql(u) \rightarrow pre(TenantID) \cup Mapping(tenantID(userID), sql) \cup post(TenantID)$, where *pre(TenantID)* is the pre personalized operation, *post(TenantID)* means the post personalized operation, and *Mapping* is the transformation function. The current tenant account is added to the Request Session with some minor modifications.

3.1.3 The SQL Request Router

The SQL Request Router sends the SQL request to different nodes of data cloud on average. One of the more powerful features is the ability to do "Read/Write Splitting". The *read* request is assigned to the *slave* node, while the *write* request is assigned to the *master* node. Database replication enables data from the master to be replicated to one or more slaves.

Replication is based on the master server keeping track of all changes to its databases (change of structure, updates, deletes, and so on) in its binary log. The binary log serves as a written record of all events that modify the database structure or content (data) from the moment the server is started. Typically, SELECT statements are not recorded because they modify neither the database structure nor content. The binary log is the collection of SQL statements after the dump operation.

3.2 Tenant Expression

Expression is a dynamic value, which is substituted when running. Common types of expressions are constants, session variables, the return value of static function and the requested variables. The tenant expression is also provided to obtain the current tenant information from the context of Web request. The syntax of tenant expression is shown in Table 1, which is used in extensible business component models to describe the personalized business.

Table 1
Tenant expression

Name	Definition
$T{tenant.tenantID}	ID of current tenant
$T{tenant.userID}	ID of current user
$T{tenant.loginName}	Login name of current tenant

3.3 Extensible Business Component Model-supported Multi-Tenancy

A **model** is a 2-tuple: *model:=(name, attributes)*, where *attributes* is a set of properties of this model, denoted as *attributes={x|x Attribute}*. The property of a model is defined as *Attribute :=(name, type, default)*, which includes a lifetime identifier, its type and the default value. We use *m(s)/f* to denote a model *m* of the system *s* in the formalism *f*. The formalism of a model is usually called a **metamodel**. The instance of a model is called an object. ***Meta****(o,m)=true* means that model *o* is the instance of metamodel *m*.

Extensible business component model (*xBC*) is proposed to describe the SaaS business to the greatest extent. The metamodel of *xBC* is divided into three different layers, shown in Figure 3: business process, business object and business presentation. A separation of design concerns into distinct model layers has several advantages, such as ease of maintenance, orientation to the viewpoint, and the ability to select specialized tools and techniques for specific concerns.

3.3.1 The Business Process Model

The business process model describes the basic business logic of an SaaS application, including *create*, *read*, *update* and *delete* (CRUD) business, compound CRUD business and user defined special business. Generally, clicking the button or hyperlink of SaaS applications in the user interface triggers the specific business process. The input of the SaaS application is often a user's form. The submission of a Web form is always triggered by a button [21]. The derived models of business process contain database-related manipulation, Uniform Resource Locator (URL), code blocks and so on. Database-related manipulation is

a direct operation of the database, such as Structured Query Language (SQL) statements and stored procedure; URL means a navigation of a Web page, such as an HTML page and JSP. Not all the business behavior can be represented in models. Some business processes are easy to describe by the source codes. Therefore, we propose a novel derived code model named *CodeBlock* which uses *dependency injection* [22] and *method interception* [22] techniques to embed source codes into models.

Business logic *BP* is defined as the instance of metamodel *BusinessProcessLogic*, satisfying ***Meta****(BP, BusinessProcessLogic)=true*. *BP* is denoted as *BP := ((name, String, ""), {(parameters, String[0..*], null), (returntype, String, "")})*.

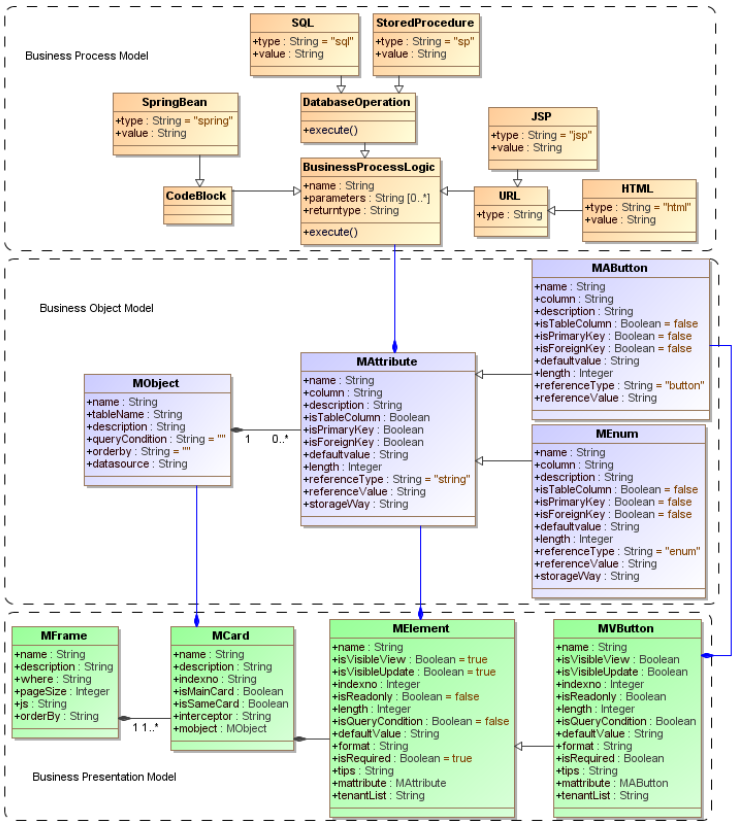

Figure 3
Metamodel of Extensible Business Component Model (xBC)

3.3.2 The Business Object Model

The business object model describes the organization of the business concepts managed by the SaaS application, which include *MObject*, *MAttribute*, *MAButton*, *Reference*, and so on. In order to refine the details of business objects, it is divided

into business object model *MObject* and the attribute model of the business object *MAttribute*. In the context of SaaS modeling, *MObject* defines the name, the description of a business object, table mappings (i.e. corresponding to the table of the relational database), and the query condition (i.e. the value range of business data represented by the instance of *MObject*). *MAttribute* describes the property of the business object, including the name, description, column (i.e. corresponding to the key of the table of the relational database), and so on. The most important property of *MAttribute* is the *reference* and *storage way*. *Reference* is made up of *reference type* and *reference value*. *Reference type* can be further broken into *primitive data type* and *special reference type*. *Primitive data type* is the data type identified by the system, such as *string*, *integer*, *Universally Unique Identifiers* (UUID) and *stringdate*. While the value of *special reference type* can be *button* (user-defined button), or *enum* (enumerated data type). These references require *reference value*, which is additional information for the reference type. *Reference value* is a series of concrete enumerated values or a list of data for the data type *enum*, and *reference value* is the name of the business process model for the data type *button*. The derived model *MAButton* is the bridge between the business process model and the business object model, which represents a special *MAttribute*. The property *storageway* of *MAttribute* presents the data storing, whether in sparse table so as to solve the SaaS "*null schema*" problem.

Business object is defined as 2-tuple, which is the instance of business object model. It is denoted as *BO :=(mobject, mattributes)*, where *mobject* is an instance of *MObject*, and *mattribute*s is a set of instances of *MAttribute*s.

3.3.3 The Business Presentation Model

The business presentation models contain the details of the graphic appearance of SaaS applications. It is composed of *MFrame*, *MCard*, *MElement* and *MVButton*. *MFrame* is the entrance to present business data for users. The instance of *MFrame* is related only to a main *MCard* and some other detail *MCard*s. Users navigate the business data represented with *MFrame* after clicking the link of the system menu. The property *where* of *MFrame* means the value range of business data in the Web User Interface (UI). *MCard* is the thinning of *MFrame* for the sake of the maintenance of a business object. The instance of *MCard* is related to several *MElement*s. The business of *MCard* is often the CRUD and other compound database business. *MElement* is the smallest unit of business presentation models, which may be the presentation of the business data. The important property of *MElement* is *isVisibleUpdate*, *isVisibleView* and *isQueryCondition*. When *isVisibleUpdate* is true, the *MElement* is a storage element. And the business data represented by *MElement* can be modified in the maintenance interface; when *isVisibleView* is true, the *MElement* is a presentation element. The business data represented by *MElement* can be only displayed in the Web UI; when *isQueryCondition* is true, it is as a query condition in the query area. These are known as *storage MElement*, *presentation MElement* and *query*

MElement, respectively. User-defined button *MVButton* is also a kind of *MElement*, and its specific business is defined in the property *referenceValue* of related *MAButton*. In order to support tenant customization, the property *tenantList* of *MElement* means the tenant list which allows the displaying of this element. Only the tenancy in the list can see the impression of *MElement*.

The business presentation object is defined as 2-tuple, which is the instance of business presentation model. It is denoted as *VO :=(mcard, melements)*, where ***Meta****(mcard, MCard)=true*, and *melements* is a set of instances of *MElement*. The Web UI object is denoted as *UI :=(mframe, vos)*, where *mframe* is the instance of *MFrame* and *vos* is a set of *VO*s.

The business presentation object is used to define the graphic UI of the business data represented by the business object model. Therefore, several basic properties of *MCard* and *MElement* of *VO*s are generated from the properties of *MObject* and *MAtttibute* of *BO*s, denoted as $m_1(s)/BO \rightarrow m_2(s)/VO$， where $BO \subset xBC$， $VO \subset xBC$. This generation is an assistant tool for modeling the details of the business presentation models, which are described in binary relation. As mentioned before, a binary relation that is specified by using a set comprehension predicate *P*, e.g., in $R = \{<a, b>|P(a, b)\}$. The values of the properties of *VO*s are generated from *MO*s according to the transformation rule r_1 and r_2, shown in Figure 4, which is further defined in the first-order predicate logic of binary relation. The rule r_1 generates *MCard*s, while the rule r_2 generates *MElement*s. The assistant tool executes the mapping rules in order implicitly.

dom $r_1=\{bo|bo \in BO\}$, where $BO \subset xBC$
ran $r_1=\{vo|vo \in VO\}$, where $VO \subset xBC$
$r_1=\{< bo, vo >|$
$(\forall bo \in BO \wedge \exists vo \in VO \wedge vo.mcard.attributes.name="C\_"+bo.mobject.attributes.name \wedge$
$vo.mcard.attributes.description=bo.mobject.attributes.description)\}$

dom $r_2=\{ma|ma \in bo.mattributes\}$, where $bo \in BO$, $BO \subset xBC$
ran $r_2=\{e|e \in vo.melements\}$, where $vo \in VO$, $VO \subset xBC$
$r_2=\{<ma, e>|$ $(\forall ma \in BO\ bo.MA \wedge bo \in BO \wedge \exists e \in vo.elements \wedge vo \in VO$
$e.attributes.name="E\_"+ma.attributes.name \wedge e.attributes.tips=ma.attributes.description \wedge$
$e.attributes.length=ma.attributes.length \wedge e.attributes.defaultValue=ma.attributes.defaultValue \wedge$
$(\forall ma.attributes.referenceType=button \wedge \exists e \in vo.elements \wedge e.attributes.defaultValue='' \ e.attributes.isQueryCondition=false \wedge e.attributes.format='') \wedge$
$(\forall ma.attributes.referenceType=integer \wedge \exists e \in vo.elements \wedge e.attributes.defaultValue=0 \wedge e.attributes.format='^-?\d+\$') \wedge$
$(\forall ma.attributes.referenceType=string \wedge \exists e \in vo.elements \wedge e.attributes.defaultValue='') \wedge$
$(\forall ma.attributes.referenceType=stringdate \wedge \exists e \in vo.elements \wedge e.attributes.defaultValue='' \wedge e.attributes.format='yyyyMMdd'))\}$

Figure 4
Model transformation rule from business object model to business presentation model

3.4 Version Control in Extensible Business Component Model

As mentioned, models are the primary artifact of the software development process in MDE. These models are typically developed by distributed environments consisting of teams at different organizations and locations. These teams usually build multiple overlapping models which represent different aspects of the same systems. In addition, models undergo a complex evolution during their life cycles. As a consequence, one of the techniques used to support model management activities is the version control of models. However, present-day MDE tools offer only limited support for the version control of models. Traditional version control systems are based on the copy-modify-merge approach [23], which is not fully exploited in MDE since current implementations lack model-orientation.

In contrast, we use Java Content Repository (JCR) [24] as the storage of models. A content repository, shown in Figure 5, consists of one or more workspaces, each of which contains a tree of items. An item is either a *node* or a *property*. Each node may have zero or more child nodes and zero or more child properties. There is a single root node per workspace which has no parent. All other nodes have one parent. The model may be considered as a node, and the property of the model may be considered as a property. The JCR 2.1 (JSR-333) [24] specification provides simple and independent versioning or full versioning of a node in the repository. A versioning repository has, in addition to one or more workspaces, a special version storage area. A new version is added to the version history of a versionable node when one of its workspace instances is checked-in. The model stored in the repository can be restored to a previous version, which is useful when developers have made some fatal mistake in modeling the system. There are two basic operations of nodes. To create a new version of a versionable node, the application calls *checkin*. In order to alter a versionable node, the node must be *checked out*. There are some open source tools fully conforming to the implementation of the JCR specification, such as Jackrabbit and ModeShape[1].

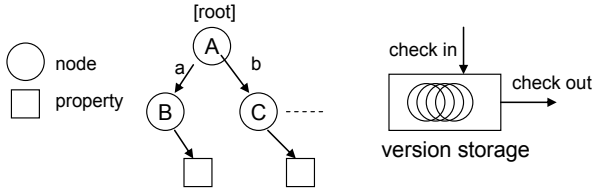

Figure 5
Java Content Repository

[1] Apache Jackrabbit and ModeShape are a JCR implementation that provides access to content stored in many different kinds of systems, which can be downloaded from http://jackrabbit.apache.org and http://www.jboss.org/modeshape respectively.

4 Template-based Model Transformation in Support of Model Synchronization

4.1 Template Engine

4.1.1 Template Data Model

The basic structure of **template data model** is a tree shown in Figure 6. The root node is the Web UI object. All the data models of SaaS applications save in the model repository.

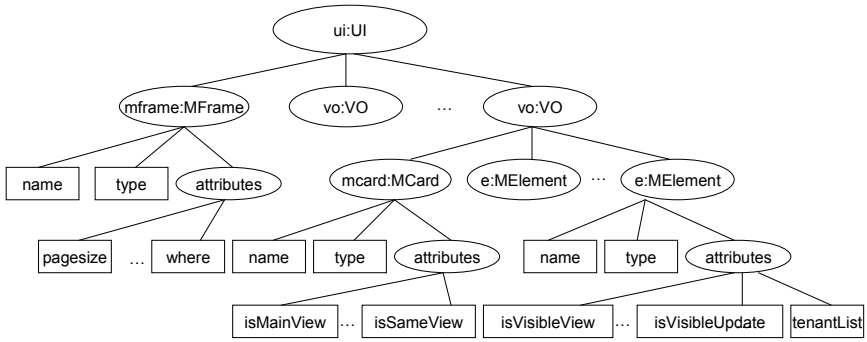

Figure 6
Template data model of xBC

4.1.2 Syntax and Semantics of Template

A **template** is a series of template statements. The set of transformation rules from *xBC* to codes is denoted as $F=\sum template_i$. The model transformation rule of the textual template evolution is based on all the template statements.

A *template statement* is defined as 4-tuple: *TemplateStatement:=<Text, Interpolation, Tag, Comment>*. The **text** is static text and it will keep constant after model transformation; the **interpolation** is used to insert the value of the expression converted to text, which is the dynamic content of templates. The format of interpolations is *${expression}*; the **tag** introduces some evolution mechanism to satisfy the requirements for the specific application field, such as macro, iteration, condition and function statements. Also the tag can execute some directives. In fact there are two types of directives: predefined directives and user-defined directives. User-defined directives are extensions of directives. Some directives have been implemented, such as Macro, Conditional directives, List directives and Function; the **comment** will be ignored and not be written to the output.

4.1.3 Template Component

The main ideas of the template reuse are to divide the templates into parts of the components, and each component can generate the relatively independent target framework. Therefore, a pluggable template called plugin, which is a series of templates, generates code based on some open source libraries (Such as SQL, Spring, Hibernate, MyBatis, Struts, JSF, Web Service, etc.).

4.2 Model Transformation from Extensible Business Component Models to Codes

Aiming to obtain software corresponding to this business system, we can use the architecture composed of Business Component (BC) and Business Process (BP). The system is an integration of composition with many business processes, one of which is connected with a series of business actions. All the BCs and BPs in the runnable system can be generated from the templated-base model transformation approach.

From the viewpoint of model transformation, the model mapping from *xBC*s to codes is denoted as $m_1(s)/xBC \rightarrow m_2(s)/Code$ shown in Figure 7. From the viewpoint of function, the model mapping is denoted as *codes=models+textual template*.

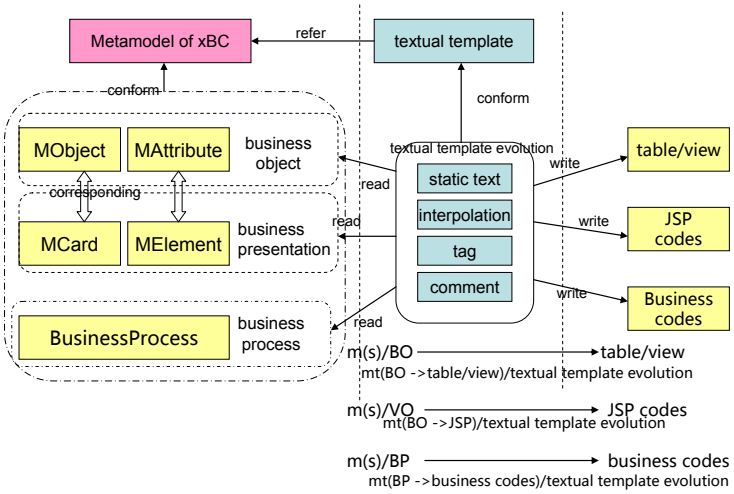

Figure 7
Transformation process between xBCs and codes

4.3 Model Synchronization Based on Model Evolution

Source models involved in model synchronization may face with the modifications shown in Table 2. The modifications in the three circumstances are identified based on the version number. All the version numbers of models involved in model synchronization are recorded. In the next model synchronization, the version of involved models is needed to compare with the last recorded version. If the model is not in the last recorded models, it is **addition**; if the new version is greater than the past and it is not a new model, it indicates that the model is **updated** after the last model synchronization; if one of the last recorded models is not involved in the next model synchronization, it indicates that the source model has been **deleted**. The model synchronization algorithm *PSM2CodeSync* from *xBC* to Web JSP codes is shown in Figure 8.

Table 2
Classification of modifications

Name	Definition
add	Source model is added
delete	Source model is deleted
update	The property of source model is changed

```
Function PSM2CodeSync
Input: ui:UI
Output: codes
// justify whether need code generation
if(isNewModel(ui)){
   PSM2Code(ui);
}elseif (isDeletedOperation(ui)){
   deleteGeneratedCodes(ui);
}else if(modelDetection(ui)){
    PSM2Code(ui);
}

Function PSM2Code
Input: mframe:MFrame
Output: codes
generateCode("query", ui);
generateCode("insert", ui);
generateCode("update", ui);
generateCode("detail", ui);
recordCurrentVersion (ui);//record the last version of models

Function generateCode
Input: templateName, ui:UI
Output: codes
helper.processText();
helper.processInterpolation();
helper.processTag();
helper.processComment();//textual template evolution
```

```
Function modelDetection
Input: ui: UI
Output: true or false
if(versionChange(ui.mframe)) return true;
for each VO vo in ui.vos {
  if(versionChange(vo.mcard)) return true;
  if(versionChange(mcard.mobject)) return true;
  for each MElement melement in vo.melements {
    if(versionChange(melement)) return true;
    if(versionChange(melement.mattribute)) return true;
    if(versionChange(melement.mattribute.referencevalue)) return true;
}}
return false;
```

Figure 8

Model synchronization algorithm between xBCs and codes

Conclusions

This study was aimed at investigating the model transformation approach to generate SaaS applications. The main contributions of the study are outlined below:

1) A data middleware of a multi-tenant database is presented. As this approach is transparent to the application, the applications of tenants can share this data model without any modification. The abstract data model provides the logic isolation of different tenants. The master/slave database in the data cloud is a kind of horizontal scalability to improve the performance of data.

2) In this paper, a novel Extensible business Component model named *xBC* is proposed for describing both the structural and behavioral properties of generic SaaS applications. The tenant expression, the property *storageWay* of *MAttribute*, and the property *tenantList* of *MElement* are presented to support multi-tenancy of SaaS applications. Its architecture of metamodel and extension mechanism is discussed in detail. In addition, we use versioning nodes of JCR as the storage of models. The model stored in the repository can be restored to a previous version according to the version number.

3) Additionally, our approach for model transformation that supports model synchronization based on model evolution is presented. This model transformation approach is based on the textual template evolution, and this model synchronization approach will only take up the storage space of model repositories rather than some extra space. Only the changed models need a subsequent model transformation. That is a good way to minimize the amount of source that needs to be reexamined by a transformation when the source is changed.

Acknowledgement

This work was supported by the National Natural Science Foundation of China under Contract Numbers 60873089 and 60903176, the Provincial Natural Science

Foundation for Outstanding Young Scholars of Shandong under Contract Numbers JQ200820, the Program for New Century Excellent Talents in University under Contract Numbers NCET-10-0863 and the Science and Technology Development Program of Shandong Province under Contract Number 2011GGX10116.

References

[1] Sánchez, P., Moreira, A., Fuentes, L., Araújo, J., Magno, J.: Model-driven Development for Early Aspects, Information and Software Technology 52 (2010) 249-273

[2] Gao, X., Li, Z.: Business Process Modeling and Analysis Using UML and Polychromatic Sets, Production Planning and Control 17 (2006) 780-791

[3] Whittle, J., Service Model-Driven Development: A Practical Approach. London: Chapman & Hall; 2012

[4] Mahowald, R P.: Worldwide Software as a Service 2011–2015 Forecast and 2010 Vendor Shares, International Data Corporation, USA, 2011

[5] Narasimhan, B., Nichols, R.: State of Cloud Applications and Platforms: The Cloud Adopters' View, Computer 44 (2011) 24-28

[6] Brunelière, H., Cabot, J., Frédéric, J.: Combining Model-driven Engineering and Cloud Computing, in Proceedings of 6th European Conference on Modelling Foundations and Applications, Paris, France, June 15-18, 2010, pp. 1-2

[7] Guo, J., Sun, W., Huang, Y., Wang, Z., Gao, B.: A Framework for Native Multi-Tenancy Application Development and Management, in Proceedings of The 9th IEEE International Conference on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, Tokyo, Japan, July 23-26, 2007, pp. 551-558

[8] Jacobs, D., Aulbach, S.: Ruminations on Multi-Tenant Databases, BTW 103 (2007) 514-521

[9] Ma K., Chen, Z., Abraham, A., Yang, B., Sun, R.: A Transparent Data Middleware in Support of Multi-Tenancy, in Proceedings of the 7th International Conference on Next Generation Web Services Practices, Salamanca, Spain, October 19-21, 2011, pp. 1-5

[10] Chang, S. H., Kim, S. D.: A Variability Modeling Method for Adaptable Services in Service-Oriented Computing, in Proceedings of the 11th International Software Product Line Conference, Kyoto, Japan, September 10-14, 2007, pp. 261-268

[11] Liu, Y., Zhang, B., Liu, G., Wang, D., Gao, Y.: Personalized Modeling for SaaS Based on Extended WSCL, in Proceedings of the 2010 IEEE Asia-Pacific Services Computing Conference, Hang Zhou, China, December 06-10, 2010, pp. 355-362

[12] Wang, Z., Zhao, Z., Fang, J., Wang X.: A SaaS-Friendly Service Community Model and Its Application in the Nationwide Service Network for Sharing Science and Technology Information, Chinese Journal of Computers 33 (2010) 2033-2043

[13] Mukerji, J., Miller, J.: The MDA Guide Version 1.0.1, Object Management Group, USA, 2003

[14] Subramanyam, R., Weisstein, F. L., Krishnan, M. S.: User Participation in Software Development Projects, Communications of the ACM 53 (2010) 137-141

[15] Egyed, A.: Automatically Detecting and Tracking Inconsistencies in Software Design Models, IEEE Transactions on Software Engineering 37 (2011) 188-204

[16] Czarnecki, K., Helsen, S.: Feature-based Survey of Model Transformation Approaches, IBM System Journal 45 (2006) 621-645

[17] Object Management Group: Meta Object Facility (MOF) 2.0 Query/View/Transformation Final Adopted Specification 1.1, Object Management Group, USA, 2011

[18] ikv++ technologies ag: medini QVT 1.7.0 (2011), http://projects.ikv.de/qvt/

[19] Giese, H., Wagner, R.: From Model Transformation to Incremental Bidirectional Model Synchronization, Software and Systems Modeling 8 (2009) 21-43

[20] Ma, K., Yang, B., Chen, Z., Abraham, A.: A Relational Approach to Model Transformation with QVT Relations Supporting Model Synchronization, Journal of Universal Computer Science 17 (2011) 1863-1883

[21] Duggan, D., Service Oriented Architecture: Entities, Services, and Resources. NJ.: Wiley-IEEE Computer Society; 2012

[22] Tanter, É., Toledo, R., Pothier, G., Noyéb, J.: Flexible Metaprogramming and AOP in Java, Software and Systems Modeling 72 (2008) 22-30

[23] Collins-Sussman, B., Fitzpatrick, B. W., Pilato, C. M., Version Control with Subversion for Subversion 1.6: The Official Guide And Reference Manual. NY.: Soho Press; 2010

[24] Nuescheler, D.: JSR 333: Content Repository for Java Technology API Version 2.1 Early Draft Review, Java Community Process, USA, 2011