Modularizing Tenant-Specific Schema Customization in SaaS Applications

Chun-Feng Liao
Department of Information Engineering and Computer Science,
Feng Chia University
cfliao@fcu.edu.tw

Kung Chen Jiu-Jye Chen
Department of Computer Science,
National Chengchi University
{chenk, 100971009}@nccu.edu.tw

Abstract

Recently, Software as a Service (SaaS), a cloud-enabled business model, has received a lot of attention in the software industry. Among various facets of SaaS, the data layer design issues, such as schema customization and schema mapping, have been identified as one of the most important challenges for realizing economically scalable multi-tenant SaaS applications. In this paper, we propose an aspect-driven approach to deal with this challenge. After presenting the motivation and rationale of our work, we sketch the overall approach based on an example SaaS application.

Categories and Subject Descriptors Information systems [*Data management systems*]: Middleware for databases

General Terms Languages, Design

Keywords multi-tenant, schema-mapping, SaaS

1. Introduction

Software as a service (SaaS) is an emerging service model of cloud computing. Its main characteristic is the ability for customers to use a software application on a pay-as-you-go subscription basis. To be economically scalable, a SaaS application must leverage resource sharing to a great extent by accommodating different users of the application while making it appear to each that they have the application all to themselves.

As well argued in [1], a SaaS application must be a multi-tenant application to catch the ”Long Tail.” In general, tenants of a SaaS application are oblivious to the fact that the resources (e.g. CPU time, network bandwidth, and data storage) are shared among tenants. Furthermore, application-level multi-tenancy [2] employs a single application instance to serve multiple tenants. On the other hand, tenant-specific application customization is also an important issue to be addressed when developing multi-tenant applications. Several attempts have been made to identify the multi-tenant concerns of SaaS applications, such as affinity (how tasks are transparently distributed), persistence design, performance isolation, QoS differentiation, and customization [2]. Koziolek obtained similar results based from the Software Architecture's points of view [3].

In particular, Cai et al. further group these concerns into two layers, that is, the application layer and the data layer [4]. In our view, any tenant-specific customizations, application layer or data layer, are cross-cutting concerns of a multi-tenant application, and thus should be modularized so that tenants are able to develop, deploy, and integrate to such customizations to the virtualized application with minimal additional programming and configuration efforts. In application layer, most concerns can be modularized by either intercepting filters [4], aspects [5, 6], or contexts [8]. On the other hand, although it is generally agreed that the multi-tenant data layer is one of the most important concerns [7], there is little investigation on modularizing data layer concerns in a multi-tenant application.

In the design space of the data layout and management strategy for multi-tenant applications, various alternative approaches form a continuum between the isolated data style and the shared data style [7]. As pointed out by Aulbach et al. [10], the shared data style provides very good consolidation but lacks schema extensibility. Much of the research on multi-tenant modularization assumes that either each tenant has a dedicated set of tables and have the same schema or all tenants are consolidated in one set of tables but share an identical schema. However, it is highly desirable to have a data layer for multi-tenant applications that can support shared storage with custom extension, namely, supporting tenant-specific schema customization using a set of tenant-shared tables.

A general approach to support such architecture is a middleware-level facility that supports the mapping of multiple single-tenant logical schemas in the application to one multi-tenant physical schema in the database. Among commonly used schema-mapping techniques, Universal Table is the most flexible one. Essentially, a Universal Table is a generic structure that has virtually no schema attached to it. Although it is commonly held that Universal Table layout would incur a large amount of performance overhead, it is the approach adopted by SalesForce.com, which is a successful SaaS vendor best known for its CRM service that supports more than 55,000 tenants [11]. It is also worthy of mentioning that many commercial databases have a ”view” mechanism for providing logical tables to applications. However, it is hard to implement tenant-specific schema customization by adapting the ”view” mechanism, since these mechanisms are typically part of DBMS and are not modifiable.

To sum up, we observe that the data layer issues of multi-tenant applications are not adequately addressed by current modularization approaches. Specifically, the schema-mapping issues are not taken into consideration, which is usually implemented in real-world multi-tenant applications such as SalesForce.com. As illustrated by Rashid and Chitchyan [9], persistence issues in the data layer of a single tenant application can be largely realized by aspects. This paper further extends that approach to multi-tenant ap-

AOAsia'13, March 25, 2013, Fukuoka, Japan.

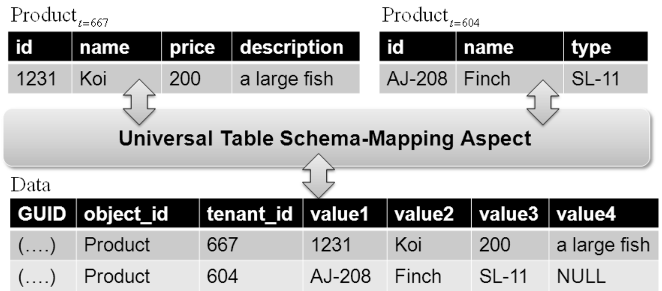

Figure 1: The mapping from logical schema to physical schema

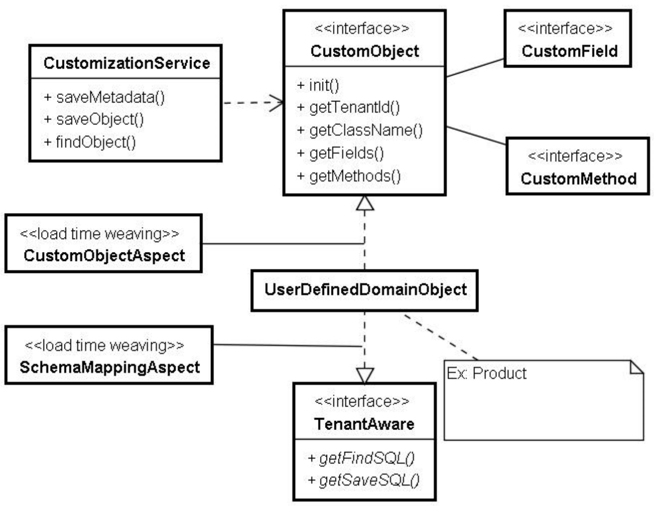

Figure 2: The structure of key components and aspects

plications with two additional cross-cutting concerns. The first one is support of tenant-specific customization of data schema, and the second one is realization of the schema-mapping logic that maps tenant-specific schema to the underlying Universal Table schema.

2. Approach

This section sketches our approach with a hypothetical multi-tenant application, namely, the ShoppingForce.com (see Fig. 1). We begins with a partial description of the schema layout of the example. Note that, in the following discussion, we follow the convention in [11] and use the term objects and tables as well as fields and columns interchangeably. The physical schema of ShoppingForce.com is similar to that of Force.com. There is a *Data* table served as the Universal Table that stores all tenants' data. To facilitate schema mapping, there are additional tables such as *Object*, *Fields*, and *Relationships* to keep track of meta information of logical tables, logical columns, and relationships, respectively.

The hypothetical application enables its tenants to sell products and to process orders on-line. Since different tenants have their own unique needs in describing their products, ShoppingForce.com allows its tenants to create their own customized schemas for their products. As illustrated in Fig. 1, initially, tenant 667 defined an logical table with object_id='Product', denoted $Product_{t=667}$, and then tenant 604 also defined another logical table with the same object_id ($Product_{t=604}$). The data in the two logical tables are physically stored in the same table, namely, the *Data* Table.

```
public aspect CustomObjectAspect {
    declare parents : ...  implements CustomObject;
    public String CustomObject.getTenantId() {...}
    public List<CustomField> CustomObject.getFields() {...}
    ...
}
public abstract aspect SchemaMapppingAspect {
    declare parents : ...  implements TenantAware;
    public String  TenantAware.getFindSQL(...) {...}
    public String  TenantAware.getSaveSQL() {...}
    ...
}
```

Figure 3: Code fragments of aspects

Fig. 2 displays the structure of the key components and aspects of our approach. In the remainder of this section, we illustrate how these components and aspects work to support tenant-specific schemas of 'Product' . First, a tenant may declare his own version of the *Product* class, which will correspond to a *UserDefinedDomainObject* in our approach. Then, the class needs to be transformed into a generic form so that it can be managed by the *CustomizationService* (see Fig. 2). Our approach will perform load-time weaving against the class (i.e. *Product*) by introducing *CustomObject* and *TenantAware* interfaces, related fields and methods. In this way, the *CustomizationService* is able to process customization and schema-mapping tasks. The code fragments of *CustomObjectAspect* and *TenantAware* are shown in Fig. 3. It is worthy of mentioning that load-time weaving is necessary since in practice it is impossible to restart the SaaS application every time when a tenant defines a new custom object. Besides, in the *CustomObjectAspect*, there must be an advice being woven into the class initialization join point of the custom object so that *CustomizationService* has a chance to maintain meta information stored in the physical tables such as *Objects*, Fields, and Relationships. After the structure of *Product* being transformed, the application is able to process the schema-mapping tasks transparently to the tenants by utilizing methods injected by *SchemaMappingAspect*. These aspects essentially introduce *CustomObject* and *TenantAware* interfaces as well as related method implementations. The type pattern between *declare parents:* and *implements* can be specified by an annotation and a wildcard as follows

```
declare parents: @Persistent *
                 implements TenantAware;
```

To illustrate how *SchemaMappingAspect* performs schema-mapping between physical and logical tables, let us assume that one of ShoppingForce.com's tenant, whose id is 667, has submitted a query statement:

```
SELECT id, name, price
FROM Product
WHERE id = '1231'
```

The request is intercepted by *SchemaMappingAspect*, and then delegated to the *TenantAware.getFindSQL* method. Next, the *TenantAware.getFindSQL* transforms the submitted logical query statement into one or more physical query statements. In this example, the submitted statement can be represented in the following alge-

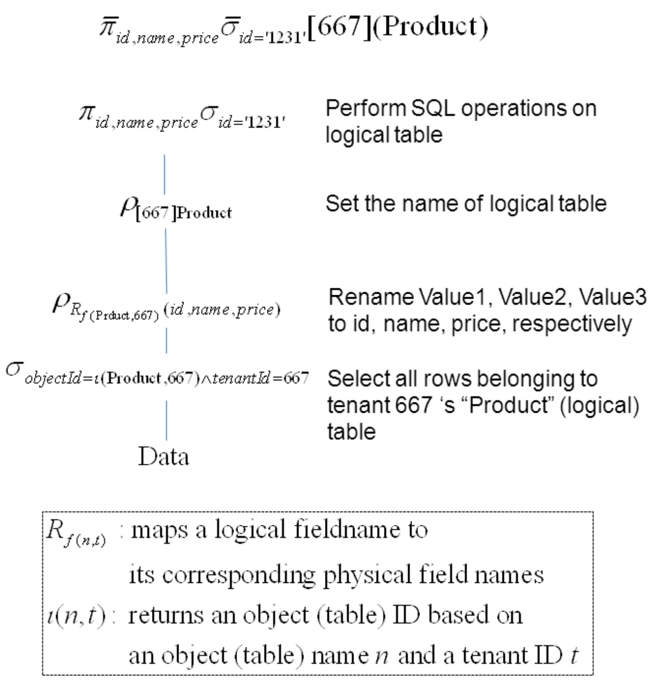

Figure 4: Query plan for the sample query statement

braic form:

$$\bar{\pi}_{id,name,price}\bar{\sigma}_{id='1231'}[667](\text{Product}),$$

where $\bar{\pi}$ means a logical projection, $\bar{\sigma}$ means a logical selection, and $[667]\text{Product}$ denotes the logical *Product* table belonging to tenant 667. Based on the algebraic expression, a query plan can be derived base on pre-specified rewriting rules [12]. For instance, Fig. 4 is the query plan for the expression mentioned above. Note that the functions $R_{f(n,t)}$ and $\iota(n,t)$ are reusable stored procedures which also involve a set of query statements. The transformed algebraic expression is listed below:

$$\pi_{id,name,price}\sigma_{id='1231'}$$
$$\rho_{R_{f(\text{Prduct},667)}(id,name,price)}$$
$$\sigma_{objectId=\iota(\text{Product},667)\wedge tenantId=667}(Data),$$

where ρ is renaming function. As a result, the expression can be transformed into the following (physical) SQL statement and return to the caller:

```
SELECT id, name, price
FROM ((
   SELECT value1 AS id,
          value2 AS name,
          value3 AS price
   FROM Data
   WHERE object_id='Product' AND
         tenant_id='667'
 ) AS Product)
WHERE id='1231'.
```

Finally, we we should point out that *SchemaMappingAspect* is an abstract aspect, since different schema layout requires different schema-mapping logics, see [10] for further discussion on various schema layout approaches.

3. Conclusion

In this paper, we present a preliminary analysis and results of a tenant-specific modularization of data schema customization for SaaS application that includes the customization of domain objects and its mapping to the underlying physical schema. To validate the feasibility of the proposed approach, our next step is to implement the proposed approach as a data management framework based on an aspect tool that supports load-time weaving, such as AspectJ or a bytecode rewriting tool.

Acknowledgments

This work is partially sponsored by National Science Council, Taiwan, R.O.C., under grant NSC-101-2218-E-035-008 and NSC-101-2627-E-002-002.

References

[1] F. Chong and G. Carroro, "Architecture Strategies for Catching the Long Tail," available at: `http://msdn.microsoft.com/en-us/library/aa479069.aspx`, accessed July 26, 2012.

[2] R. Krebs, C. Momm, and S. Konev, "Architectural Concerns in Multi-Tenant SaaS Applications," in Proc. International Conference on Cloud Computing and Service Science (CLOSER12), 2012.

[3] H. Koziolek, "The SPOSAD Architectural Style for Multi-tenant Software Applications," in Proc. 9th Working IEEE/IFIP Conferences on Software Architecture, pp. 320-327, 2011.

[4] H. Cai, N. Wang, and M. J. Zhou, "A Transparent Approach of Enabling SaaS Multi-tenancy in the Cloud," in Proc. IEEE World Congress on Services, 2010.

[5] H. Wang and Z. Zheng, "Software Architecture Driven Configurability of Multi-tenant SaaS Application," in Proc. International Conference on Web Information Systems and Mining, Lecture Notes in Computer Science, Vol. 6318, pp.418-424, 2010.

[6] H. Shimamura, K. Soejima, T. Kuroda, and S. Nishimura, "Realization of the High-density SaaS Infrastructure with a Fine-grained Multitenant Framework," in NEC Technical Journal, Vol. 5, No. 2, 2010

[7] F. Chong, G. Carraro, and R. Wolter, "Multi-Tenant Data Architecture. MSDN Library," http://msdn.microsoft.com/en-us/library/aa479086.aspx, accessed 2012.12.29.

[8] E. Truyen, N. Cardozo, S. Walraven, J. Vallejos, E. Bainomugisha, S. Gunther, T. D'Hondt, and W. Joosen, "Context-oriented Programming for Customizable SaaS Applications," in Proc. ACM Symposium on Applied Computing, pp. 418-425, 2012.

[9] A. Rashid and R. Chitchyan, "Persistence as an Aspect," in Proc. ACM SIGPLAN International Conference on Aspect-oriented Software Development (AOSD'03), 2003.

[10] S. Aulback, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger, "Multi-Tenant Databases for Software as a Service: Schema-Mapping Techniques," in Proc. ACM International Conference on Management of Data, 2008.

[11] C. D. Weissman and S. Bobrowski, "The design of the Froce.com multitenant internet application development platform," in Proc. ACM International Conference on Management of Data, 2009.

[12] C. F. Liao, K. Chen, and J. J. Chen, "Toward a Tenant-aware Query Rewriting Engine for Universal Table Schema-Mapping," in Proc. of the IEEE International Conference on Cloud Computing Technology and Science (IEEE CloudCom 2012), presented in 2012 International Workshop on SaaS (Software-as-a-Service) Architecture and Engineering, Taipei, Taiwan, 2012.