17th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2012), Paris, France, July 2012, IEEE CS Press.

SMURF: Supporting Multi-tenancy Using Re-Aspects Framework

Mohemed Almorsy and John Grundy
Computer Science & Software Engineering, Faculty of Information & Communication Technologies
Swinburne University of Technology, Hawthorn, Victoria, Australia
[malmorsy, jgrundy]@swin.edu.au

Abstract — **Software-as-a-service multi-tenancy helps service providers to cut cost, improve resource utilization, and reduce service customization and maintenance time as the tenants share the same service instance. However, existing large-scale business applications inherently do not support multi-tenancy. This hinders these applications' vendors from adopting the cloud model. Thus reengineering such applications to support multi-tenancy has become a key requirement. Reengineering such applications to support multi-tenancy is a complex and challenging task as it requires a deep understanding of the given application and almost all system modules need to be revisited. In this paper we introduce SMURF, Supporting Multi-tenancy Using Reengineering Aspects "Re-Aspect" Framework, to help service providers reengineering their applications to support multi-tenancy. SMURF is based on our new re-aspects concept. A given system modification including code to be disabled, modified, replaced or injected is encapsulated in a re-aspect. SMURF realizes given system modifications by automating the change impact analysis process as well as the change propagation process. We analyze the multi-tenancy pattern, discuss the set of requirements to migrate a single-tenant application to support multi-tenancy, describe SMURF approach, architecture and implementation. We discuss our evaluation experiments of SMURF using a set of open source web applications.**

Keywords: **Cloud Computing; Software-as-a-Service; Multi-tenancy Reengineering; Re-Aspects**

I. INTRODUCTION

Cloud computing [1] is a new paradigm shift in computing platforms with an emphasis on increasing business benefits. The cloud model is leading the IT industry towards new service delivery models based on service outsourcing and the pay-as-you-go payment model. Customers can rent services occasionally and pay only for amount of resources they use. Software-as-a-service (SaaS) [2] is one of the key service deliver models delivered by the cloud computing. SaaS helps reducing infrastructure, license and administration cost. This helps servicing small and medium enterprises (SMEs) "long-tail-market" as well as mega customers at the same time probably on the same service instance using multi-tenancy pattern.

Multi-tenancy helps in delivering services including infrastructure, platform and software that can be shared between different tenants. In the IaaS model, multi-tenancy is achieved through using hypervisors that virtualize the server resources. Thus the OS does not need to be changed. Moreover, each customer has a separate instance (VM). However, In the SaaS model, multi-tenancy has different possible deployment models including a separate instance for each tenant up to a single instance for all tenants. The later deployment model is definitely the optimal model. However, it requires the SaaS application to handle multi-tenancy and tenants isolation itself.

Supporting multi-tenancy requires the SaaS application to support capturing, processing and storing data of different tenants in the same application instance. Moreover, the SaaS application should maintain security and performance isolation between its tenants. This requires considering multi-tenancy as a key requirement from the early stage of the system development process. Many of the existing well-known, large-scale business applications that are widely used nowadays are locked-in a high cost business model. This prohibits them from targeting/servicing the "long-tail-market". Thus, it becomes a business need to migrate such applications to support multi-tenancy. Migration of such applications to support multi-tenancy is a very complicated task as it requires a deep understand of the application. Moreover, there are lots and lots of system modifications are required to be delivered. This requires revising/updating almost all system modules.

Existing efforts to support multi-tenancy either focus on extending applications to support multi-tenancy by wrapping a single-tenant application with a platform that manages the multi-tenancy dimension [3-8]. The same approach has been followed in industrial efforts as well [9]. Using these approaches, applications are locked-in to cloud platforms that have such multi-tenancy platform hosted. Moreover, features such as user interface customization, application model extension, etc. will not be available for tenants if the original applications do not support them. Limited efforts targeted conducting real reengineering of applications to support multi-tenancy [10, 11]. These efforts focus only on providing a systematic process to be followed by system engineers to support multi-tenancy. It does not have tool support to automate this process.

We introduce SMURF, a novel application reengineering process and approach, along with a tool support, to support multi-tenancy. Our approach is based on reengineering-aspects "Re-Aspects" concept where a given system modification (change request) is captured as a re-aspect. The re-aspect includes the signature of code snippets

to be modified, action to be applied on identified locations (instances of these signatures) include inserting, modifying, replacing, or deleting code parts, and code to modify, replace or insert. We conduct a thorough analysis of the multi-tenancy pattern for web applications and come up with a set of key requirements/modifications that should be addressed at the database, data access layer, business logic layer, and presentation layer levels. Moreover, we studied the requirements for security and performance isolation. We developed a set of modification patterns that the system engineers can use when reengineering applications to support multi-tenancy. Given these modifications specified as re-aspects, SMURF analyzes the system source code to identify code snippets that match the specified re-aspect's signatures, and then it performs impact analysis to identify the entities that need to be modified to realize the given modification. Finally, SMURF use the specified re-aspect actions to update the identified matches. We have implemented a prototype for SMURF and tested it in reengineering an ERP system, developed internally, to support multi-tenancy. We have evaluated our approach in reengineering a set of open source web applications to support multi-tenancy.

The paper is organized as follows: section II shows a motivating example for our research problem. Section III reviews related work in the area of reengineering SaaS to support multi-tenancy. Section IV introduces an analysis of the SaaS multi-tenancy reengineering requirements. Section V gives an overview on Re-Aspects. In section VI, we discuss the implementation of our approach. Section VII shows the results of evaluating our approach in reengineering a set of open source applications. Section IX explains the implications of our work and further research.

II. Motivation

Consider SwinSoft a software house that has an existing web-based ERP system called Galactic. Galactic is developed using C# and ASP.net. It is currently used by a noticeable number of customers. SwinSoft is attracted by potential market share of SaaS. Thus they has decided to migrate Galactic to the cloud (to support multi-tenancy) to address the long tail market customers and increase their return of investment.

Galactic delivers a set of modules including sales management, human resources management and purchasing. Its architecture is made up of a database that has more than 100 tables along with a set of views and stored procedures. Some of these tables are expected to maintain huge number of records including customers, invoices, payments and returns tables. The data access layer developed using NHibernate; a business logic layer including business objects, business rules and workflow; and presentation layer developed using ASP.net. Data lookups are retrieved using stored procedures directly without using business objects.

SwinSoft has conducted a preliminary analysis to identify the requirements to address to support multi-tenancy. The key requirements identified from this analysis include securely isolating different tenants' data, supporting extensible database schema (tenants can define their own fields), user interface customization and branding per tenant, different workflow and business rules based on current tenant, etc. On the other hand, SwinSoft is suffering from a high turnover rate. Moreover, the documentation of Galactic is outdated. It is highly required to lunch Galactic-for-Cloud as soon as possible to take part in the market share.

III. Related Work

The area of multi-tenant cloud applications is relatively new. Moreover, a limited number of these efforts focus on reengineering applications to support multi-tenancy as a new architectural pattern. Most of these efforts target introducing system wrappers to adapt existing single-tenant applications to support multi-tenancy without modifying the target system itself. Bezemer et al [12] discuss the possible challenges in migrating a single-tenant application to support multi-tenancy. This includes performance, security, scalability, Zero-Downtime, etc. They also propose a new blueprint of a SaaS platform [3] that can extend a single tenant application to support multi-tenancy. The platform wraps the system and extends its authentication and configuration capabilities to support multi-tenancy. Moreover, the platform has a filtering component that stands between the system and the database. It adds "tenantID = X" filter to every query sent to the database. Although, the platform requires limited modifications, it depends on the target system delivered features. Thus if the system does not support defining custom fields for example, the multi-tenant version will not support it. Moreover, it has a performance overhead as every transaction has to pass through the platform first. Hong Cai et al [4, 5] propose a transparent approach to transform existing web applications into multi-tenant SaaS applications. They intercept Web requests to the target system and derive the tenant context, carry the tenant context with a thread in the Web container, manipulate the isolation points, and propagate tenant context to application resources. Chang Jie Guo et al [6] developed a multi-tenancy enabling framework. The framework is based on a set of common services that provide security isolation and performance isolation. Their security isolation pattern considers the case of different security requirements (for authentication and access control only). However, it depends on the tenant's administration to manually configure security policies, map tenant's users and roles to the application predefined roles. Dunhui et al [7] propose an architecture for cloudification of legacy applications. The architecture consists of three parts: a Web portal, a SaaS service supermarket, and a SaaS application development platform. The web portal and the Saas application development platform are fixed components will the SaaS service supermarket is used to register a given legacy system. IBM [9] introduce application reengineering process along with a multi-tenant server to enable applications to support multi-tenancy without re-engineering. Application

requests to access the database are passed to an abstract database that can append filters for requests based on the requesting tenant. Application configuration is supported by a SaaS Cockpit.

Xuesong et al [10] introduce a systematic process to extend applications to support multi-tenancy. They focus on data model, access control and tenant management aspects. They define a target multi-tenant application architecture model and define gaps between the existing legacy application and the target model. For each requirement they describe what the service providers need to do to meet the specified requirements. No tool support to help service providers in identifying or realizing the possible gaps.

IV. Analysis of Multi-tenancy Re-engineering Requirements

Before we explain how to migrate applications to support multi-tenancy using SMURF, we introduce an analysis of the possible modifications that may be required when reengineering an application to support multi-tenancy. We project our analysis to a typical architecture of a web application explaining what need to be added, modified, replaced, or deleted during the process of supporting multi-tenancy. Service providers have to make their own decisions based on their application architecture and the multi-tenancy paradigm they plan to adopt.

A. Multi-tenant Data Model

The multi-tenant application's database has different possible architecture models. The service provider has to select between these architecture models based on the isolation level they plan to deliver, scalability of the application, number of tables, expected sizes of the data tables and impact on system performance, and many other factors. This architecture models include [13]:

*1. Separate database***.** In this model, the service provider maintains a separate database for every tenant. This represents the highest level of isolation of the three models. Moreover, this model is the easiest model for migration. On the other hand, database servers are usually limited in number of databases that they can host.

2. Shared database ***but*** *separated schema***.** In this model, all tenants share the same database but different schemas. Thus each tenant has his own set of tables grouped under one schema. This provides a logical isolation between tenants' data. This approach has a problem with database restore. Moreover, this approach is suitable for applications that have small number of tables. It mitigates the limitation arise from using separate database.

3. Shared database ***and*** *shared schema***.** In this model, all tenants share the same database and the same schema. Although this model is considered the most cost effective solution, it highlights the isolation problem between tenants' data. Developers have to make sure that every tenant cannot access other tenants' data. This may require modifying the whole application to consider passing the tenantID in every query. This requires an intensive analysis of every system function. A simple common solution to this problem is to use database views to perform the filtering task. This in turn requires modifying all database queries to work on views-level not tables. To save these efforts, developers can rename all tables to be (initial + table name) – e.g. rename table Employees to sys_Employees. Then create views for all tables with the original table name – e.g.

```
CREATE VIEW Employees AS
SELECT * FROM sys_Employees WHERE TenantID = USERID
```

Thus all application queries will be projected on views where data are already filtered by the current tenant. Of course, this requires using security impersonation when application connects to the database server. Another possible option is to modify the database connection in the data access layer so that all requests are redirected to a proxy where queries are validated and filtered before submitted to the actual database.

4. Mix Model. In this model, the service provider support different data models and leave it to the tenant to select the model that best fits his needs (scale, security…).

Another issue that should be considered is how to enable data model extensibility – i.e. every tenant may have special fields or data items that they need to maintain for every operation (record). This is straight forward when maintaining separate database or separate schema per tenant. However, it still has to be propagated to the next layers. There are different approaches to realize the data model extensibility including: *pre-allocated fields* where service providers define a set of dummy columns in every table they expect that their tenants may need to extend; *Name-value pairs* where the service provider define one or more table to maintain other tables extensions. This table structure will look like tableID, tenantID, attributeName, and attributeValue; and *XML extension column* where every table has a predefined column of type XML where tenant extension columns can be maintained as one entity that can be saved and loaded.

The selection of the database model and the model extensibility to adopt, impacts the modifications required in the next layers/tiers.

B. Data Access and Business Logic Layers

In these layers we need to modify public methods' signatures to expect tenantID as a parameter. Methods' bodies should be modified as well to process the tenantID – e.g. adding tenantID to database queries, file access commands, database connection strings, loading business rules, loading and initializing workflow engine based on current tenant. The tenantID is usually propagated from the presentation layer to these layers. Both the data access layer and the business logic layer should handle custom fields (data model extension) based on the model adopted in the database. This includes how to load, store, and query these fields. Business objects' classes should be modified to include data members for tenant's information.

C. *Presentation Layer*

This layer has a set of potential modifications including user interface customization and branding (e.g. company logo, styles, themes, etc.), adding TenantID to session state, modifying calls to business logic layer functions to pass tenantID, modifying used business objects to set/get tenantID. Moreover, the presentation layer should support displaying different custom fields based on the current requesting tenant.

D. *Non-Functional Requirements*

Multi-tenant application has to be scalable to support the potential number of tenants and their workloads. This requires the service providers to deploy their application using web farms and clusters. SaaS SLAs usually capture tenants' security, performance, availability, etc. requirements that should be satisfied by the SaaS application. This resulted in issues related to how to maintain performance isolation where the execution flow of a given tenant should not be impacted by other tenants'. Load balancers with performance controllers can help in solving this problem. This requires applications to be stateless. SaaS applications should maintain session information either on the client side or on a shared server that is accessible to all other servers in the cluster.

Security is another nonfunctional requirement that should be addressed. A SaaS application should support customizing applications to support tenants' security "tenant-oriented security". This can be achieved by externalizing the security from the multi-tenant system by calling a standard library that performs authentication, authorization, etc. based on tenants' requirements and security controls. This enables every tenant to use his security controls – e.g. to use his LDAP server to authenticate and authorize users.

E. *Metadata services*

This is a key module in a multi-tenant application. It helps tenants and service providers to customize (branding) the application to match tenants' business needs. This includes customizing the user interface text, fields, visibility, and security capabilities, customizing the business workflow and business rules.

F. *Tenant On-boarding (Tenant Provisioning System)*

The registration of a new tenant should be managed by a separate tenant administration service (may be an extension of an existing system administration module). This includes batches to restore a new database instance of the system template if the "separate DB" model is applied, or create a new schema with necessary tables. This module should also enable specifying security permissions for users, roles, screens and controls. It may also include screens' customization and localization.

Table1 shows a summary of the modifications required in reengineering a given system to support multi-tenancy.

V. RE-ASPECTS OVERVIEW

The reengineering aspects "Re-Aspect" is a new concept inspired from the AOP in order to effectively support system re-engineering and maintenance domains. In re-engineering, an existing target application has code scattered through it that we need to remove, replace, modify or add additional code to, in order to effect the desired changes. In effect, we want to identify such code blocks, sometimes replacing them, sometimes selectively modifying, and sometimes inserting new code into them. Such code blocks can be coarse-grained (classes and methods) or fine-grained (lines-of-code). Moreover, these blocks may have different formats, structure or even written in different languages. This leads us to the concept of re-engineering aspects, or "re-aspects". A Re-aspect is analogous the "aspect" in traditional AOP excepting that we want to apply modification actions on the matched source code blocks in the target system.

A re-aspect specifies an atomic modification to be applied on the target system. Figure 1 shows the re-aspect definition grammar. Every re-aspect should have signature, action, and an advice. A *re-aspect signature* defines the target system entities that should be deleted/modified/replaced or into which new code is inserted – may be a line of code or declaration, whole method, or class. The signature should specify the signature type and the signature expression. A re-aspect *instance* is a system entity that matches a given re-aspect signature. A re-aspect *action* specifies what to do in every identified re-aspect instance. The action should specify the action type and conditions, if any. The advice specifies code to replace or inject or the code used to modify the existing code.

```
Re-aspect Definition  ::= s:{Signature} a:{Action} d:{Advice}
           Signature  ::= st:Signature Type se:{Signature Expression}
      Signature Type  ::= code-snippet | ocl-expression
              Action  ::= at:Action Type ac: {Action Condition}
         Action Type  ::= Delete | Modify | Replace | Inject
    Action Condition  ::= ocl-expression
```

Figure 1: Re-aspect Grammar

Based on the re-aspect action type, we have four possible re-aspects types: *Adding re-aspect:* this is a conventional aspect. Code to be injected is specified in a separate advice that is weaved with the target system at a given re-aspect instance. It has more capabilities to add any static structure to system entities. *Deletion re-aspect "anti-aspect":* this re-aspect has only signature and no implementation. The identified code blocks - re-aspect instances - are removed from the target system. *Replacing re-aspect:* this aspect is a combination of deletion and adding-aspect. It includes signature of code to be removed and an advice to be injected. *Modifying re-aspect:* this is the most complicated re-aspect. It makes use of the identified re-aspect instance code to allow the aspect developer to specify selective deletion, reordering, or addition of new nodes into the identified code instance.

Supporting system reengineering requires a powerful signature specification approach. Re-aspect supports hybrid

approach based on a flexible syntactical code snippet signature specification approach and a semantic signature specification approach.

```
1  //update namespace name for specific namespace, if any
2   namespace DummyNamespace {
3  // update Class name for re-aspects on specific class
4     class DummyClass {
5           // update method modifier, return type or
6           // name for specific method signatures
7         public void DummyMethod() {
8            DummyStatement;
9            // update method body in case of code block re-aspect
10           if (DummyCondition) {
11           }
12           for (dummy1; dummy2 ; dummy3) {
13           } } } }
```

Figure 2: code snippet re-aspect template

The *Syntactical Code Snippet Pointcut Designator* (Fig2) helps developers to specify a flexible code snippet as a re-aspect signature. Developers can specify in which namespace, class, or method this code snippet signature should exist or they can leave it dummy which means that any class, or method that matches the code snippet. The same applies for method body statements.

A	**Context** Method **inv** PublicMethods: self.IsPublic = true
B	**Context** Method **inv**: self.Body.Contains(stmt:InvocationExpression | stmt.Method.Name = "XYZ")

Listing 1: Sample OCL-based re-aspect signatures

The *Semantic Pointcut Designator (OCL-based) approach (Listing 1)* supports more formal and semantic re-aspect signatures. It uses the Object Constraint Language (OCL) as a signature definition language. It is easier, familiar for developers. Moreover, it is extensible and formal. To support specifying an OCL constraint, we developed a system-description class diagram capturing entities that exist in a given system including component, class, instance, method, inputs, and input sources. This model is used as a reference to validate developers' specified OCL signatures.

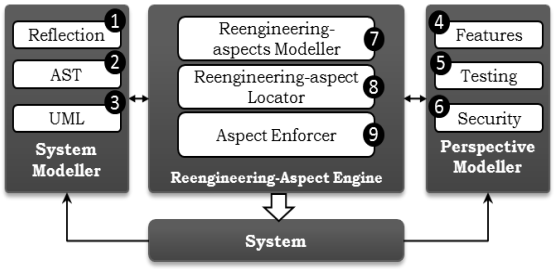

Figure 3: Re-Aspects system reengineering Framework

VI. SMURF: Supporting Multi-tenancy Using Re-Aspects Framework

The architecture and process of reengineering an application using re-aspects framework, Figure3, goes as follows:

Build the Target System Model – This is automated using reverse engineering techniques either applied on system binaries (Reflection) or system source code (using language parsers). If the system models already exist (UML), we can use them directly.

Model System Perspectives - This step is very crucial in case we are interested to get SMURF to help in specifying re-aspects signatures that take into account system perspectives rather than source code – e.g. get all methods that realize a given feature such as workflow engine. Moreover, it helps in extending the change impact analysis of a given change request to include system perspectives – e.g. test cases, features, security as well as source code entities.

Model Re-aspects – System engineers model system modifications they want to carry out on the target system. For each required system modification, system engineers specify a set of re-aspects that delete, replace, modify, or insert code at different places based on a given re-aspect signature. Fig4 shows a sample of a re-aspect definition. This is a re-aspect to modify business layer methods and add a tenantID parameter. The re-aspect signature type is (OCL), the re-aspect signature is (get all methods in the business layer). The action type to apply is (modify), the action condition is (methods that do not have parameters of business-object which should have the tenantID encapsulated inside the given business object. The advice to apply is to modify the method signature by adding more parameter called TenantID with type string. Table2 shows a summary of possible system modifications re-aspects' signatures for modification shown in Table1.

```
1   s:{   st:OCL
2         se:{
3            Context Method inv: self.Contains(s : MethodDeclaration |
4                        s.Namespace.Contains("BusinessLayer") )
5         }   }
6   a:{   at:{Modify}
7         ac:{self.Params.OCLType().BaseClass <> "BusinessObj" }   }
8   d:{
9        self.Params.Add(new Parameter("string", "TenantID")
10    }
```

Figure 4: Re-aspect Instance

Locating Defined Re-aspects - Given a re-aspect definition, SMURF checks the aspect signature type first. If it is a code snippet it traverses the code Abstract Syntax Tree (AST) looking for matches to the given re-aspect signature. If it is OCL-based, it generates a Visitor class to implement the specified OCL constraints to be used while traversing AST nodes. Fig5 shows a snapshot of the UI of the re-aspect locator. We developed this locator to be used in testing purpose while developing re-aspects' signatures, we can use this tool to make sure that the specified signature retrieves the expected instances.

Re-aspects Enforcement – Given the identified instances of the current re-aspect, SMURF executes the specified actions for the given re-aspects including injecting code (adding re-aspect), removing code (anti-aspect), replacing code (replacing re-aspect), or executing the aspect code (modifying-re-aspect). The Aspect Enforcer propagates changes on source code as well as the system class and perspectives' diagrams. Then it compiles the resultant code to make sure that no compilation errors have been introduced. The code injected by the enforcer depends on the target system entity language not on the aspect language.

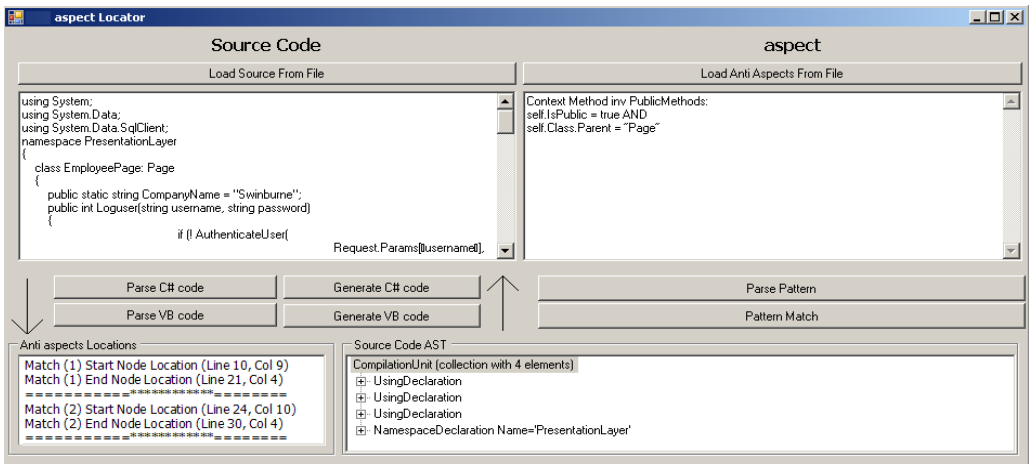

Figure5: Re-Aspect Locator

VII. USAGE EXAMPLE

Here we demonstrate how the service provider can use SMURF in reengineering their applications to support multi-tenancy. We use the motivating example from section II.

1. Define change requests' and signatures: SwinSoft system engineers should define signatures for isolation points including fields and methods. Examples of isolation-points signatures defined using re-aspects specification language are shown in Listing2. Table1 shows a full list of possible system modifications. Table2 shows a full list of their corresponding re-aspects' signatures. These tables can be used as a reference for engineers based on their needs.

Static Fields	**Context** FieldDeclaration **inv** staticFields: Self.IsStatic = true AND Self.ParentClass.IsPublic = true
Presentation Layer Methods	**Context** Method inv PublicMethods: self.IsPublic = true AND self.Class.Parent = "**Page**"

Listing 2: Samples of isolation-points signatures

2. Locating Isolation-Points: the re-aspects framework uses the specified re-aspects signatures defined by the system engineers in the previous step. The framework engine locates code snippets/entities that matches the signatures specified, as shown in Fig5.

Modify Method Signature	Method.Parameters.Add(new Parameter("TenantID", "Guid")
Inject code to extract TenantID	String currentTenatID = Session["TenantID"];
Modify Method Invocation	InvocationExpression.Argument.Add(new IdentifierExpression(currentTenatID));
Inject code to add TenantID Param	db.AddInParameter(command, "tenantId", DbType.Guid, tenantId);

Listing 3: Samples of code modifications aspects

3. Specifying Required Modifications: the next step in our reengineering process is to define modifications to be applied on every Re-Aspect instance. Static Fields may be replaced by lists or dynamic arrays, or it may be replaced by a class that reads and writes these values in configuration files based on current requesting tenant. Methods identified in the Presentation layer should be modified to extract tenantID from the current session context. And all calls to business logic layer methods should be updated by passing TenatID. Listing3 shows examples of these system modifications. Here we have different possible modifications including modify, inject, delete, and replace.

4. Applying the Specified Re-Aspects: SMURF applies the specified modifications on the identified re-aspects' instances. This results in updating the application source code – i.e. weaving specified advices (Listing3) at the identified re-aspects' instances.

VIII. IMPLEMENTATION

SMURF is implemented using .Net technology. Re-aspects details (signature, action, advice) are captured using a domain-specific visual language developed by Microsoft VS2010 modeling tool. We use .Net parsers to generate the system Abstract Syntax Tree (AST) for the given system. This helps in simplifying OCL signatures to be on abstract level not code level.

The signature locator module traverses the source code AST to locate code parts that match re-aspects captured as code snippets. If the re-aspect signature is defined as OCL expression, the signature locator generates a corresponding visitor class (a class that is used to traverse AST looking for nodes of special types and build special conditions). Methods of the visitor class are triggered whenever a match is found in the AST. The signature locator has a UI where system developers can test their signatures validity.

The weaving module is used to update the source code with actions specified in the re-aspect's definition. The procedure to follow in weaving depends on the re-aspect type. For addition re-aspect, the weaver injects the re-aspect advice source code at the re-aspects instances' locations. For deletion re-aspect, it deletes the retrieved nodes (re-aspects' instances) directly from the source code AST. For replacing re-aspect, it translates the given replacing code into AST and the deletes the AST node of the re-aspect instance and insert the new code sub-AST instead of it. For modifying re-aspects, the weaver uses the specified re-aspect advice (how to modify) as a function and pass the re-aspect instance by reference to the advice code, which can modify the passed-in AST.

Table1: List of changes required for every layer

Layer	Change Request
Presentation Layer	(1) All web pages should load layout, localization and menus based on requesting tenant. (2) Entity extension fields should be loaded based on current tenant. (3) Every page grid-view column, user control should be enabled based on user security customization for current user's tenant. (4) All business functions should receive tenantID param. (5) Set tenantID field for every business entity created in the presentation or the business logic layer. (6) All entities display pages should include the tenant defined custom fields. (7) All entity insert/edit pages should include the tenants' defined custom fields.
Business Layer	(8) All workflow definitions should filter by tenantID. (9) Update web services to have tenantID param. (10) Update all business functions to have tenantID param.
Data Access	(11) All SQL queries should filter by tenantID. (12) All Linq queries should filter by tenantID. (13) All stored procedures should have parameter tenantID. (14) All business entities should have extra attribute of tenantID.
Database	(15) Update all database tables with tenantID column. (16) Add new table for tenants' data.
QOS	(17) User Authentication and Authorization should be done through the customer security controls, if any. (18) Support Load balancing and meet tenants' SLA.

Table2: List of Re-Aspects defined in Table1 signatures

CR No.	CR Signature
1	**Context** Method **inv** loadMethods: self.Class.GetBaseType() = "Page" **AND** self.Name = "Page_Load"
2	**Context** Method **inv** fieldExtension: self.Class.GetBaseType() = "Page" **AND** self.Name = "Page_Load" **AND** self.Contains(s : IfElseStatement | s.condition = "Page.IsPostBack"
3	**Context** Method **inv** fieldSecurity: self.Class.GetBaseType() = "Page" **AND** self.Name = "Page_Load" **AND** self.Contains(s : IfElseStatement | s.condition = "Page.IsPostBack"
4	**Context** Method **inv** businessfns: self.Contains(s : InvocationExpression | s.fnName.Contains("BusinessLayer"))
5	**Context** Method **inv** businessentity: self.contains(s : newObjectStatement | s.ClassName = "businessentity")
6	**Context** Method **inv** fieldExtension: self.Class.GetBaseType() = "Page" **AND** self.Name = "Page_Load" **AND** self.Contains(s : IfElseStatement | s.condition = "Page.IsPostBack"
7	**Context** Method **inv** fieldExtension: self.Class.GetBaseType() = "Page" **AND** self.Name = "Page_Load" **AND** self.Contains(s : IfElseStatement | s.condition = "Page.IsPostBack"
8	**Context** Method **inv** wrkflwfns: self.Class.Component = "Workflow"
9	**Context** Method **inv** webservicemethods: self.Class.GetBaseType() = "Webservice"
10	**Context** Method **inv** businesfns: self.Class.Component = "BusinessLayer"
11	**Context** Method **inv** sqlqueries: self.Contains(s: InvocationExpression | s.fnName = "ExecuteScalar" OR "ExecuteQuery")
12	**Context** Method **inv** Linqqueries: self.Contains(s: QueryExpression)
13	**Context** Method **inv** sqlqueries: self.Contains(s: InvocationExpression | s.fnName = "ExecuteScalar" OR "ExecuteQuery"
14	**Context** Class **inv** businessentityDef: self.ClassName = "businessentity"
15, 16	DB script
17	**Context** Method **inv** sqlqueries: self.Contains(s: InvocationExpression | s.fnName = "Redirect" AND TargetObject = "Response" AND Arguments.Contains("Login.aspx")
18	**Context** Class **inv** sessionmgmt: self.GetBaseClassType() = "IHttpModule"

Table 3: Results of validating SMURF against .Net open source applications and its performance

System	KLOC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Time (sec)
GalacticERP	7	√	√	√	√	√	√	√	√	√	√	√	√	○	√	8
PetShop	1	√	√	√	√	○	√	√	○	○	√	√	○	○	√	5
SplendidCRM	194	√	●	√	√	○	√	√	×	○	○	√		√	○	90
NopCommerce	355	√	√	√	√	√	√	√	√	√	√	√	√	√	√	205
BlogEngine	18	√	×	√	√	√	×	×	○	○	●	√	√	○	√	15

(√) CR successfully implemented (●) CR Partially succeeded
(×) CR modification failed (○) CR is not required

IX. DISCUSSION

A. Experimental evaluation

To evaluate our re-aspects-based multi-tenancy re-engineering approach - SMURF, we have identified a set of 18 changes that should be implemented on a given web application to enable multi-tenancy, shown in table 1. We have grouped these requirements based on the layer it is related to (presentation layer, business logic layer, etc.). For each one of these changes we have developed a re-aspect signature as well as actions required to be taken for each re-aspect, shown in table 2. These signatures need to be changed slightly from an application to another. We have used SMURF to migrate a set of five .NET applications, shown in Table 3. Table 3 shows that we have successfully applied SMURF to migrate these applications. Some of these applications do not have features that we look for to change (○). SMURF failed to apply some changes on a set of given applications (×). This is specific for BlogEngine as code is mixed with html in the same file. Also SMURF has successfully implemented other changes (√). We have evaluated the SMURF performance in locating the instances to be identified for all changes defined in table1. The results of our performance evaluation are shown in table 3.

B. Threats to validaty

SMURF is based on the "re-aspects" concept, where engineers specify change requests required on a target

system to replace, insert or modify system code snippets to meet the new requirements. Re-aspects provide more customizable actions than weaving code before, after or around a system entity (AOP). Re-aspects signatures are highly flexible to capture syntactical and semantic code snippets and signatures on abstract representation rather than the system source code. Re-aspects signatures can also capture semantic signatures with dimensions other than code such as system features, architecture, design and testing entities using OCL expressions. A Key issue with the re-aspect signature is how to make sure that the developers have written the correct signature definition for target system entities they want to locate and change. This is, of course, also an issue for conventional AOP aspect specification and debugging. Currently we provide the re-aspect locator UI, Fig5, to help developers in testing their signatures. Moreover, we depend on testing to make sure that the specified modifications correctly implemented.

Code updating usually suffers from code dependency problems where the existing code depends on code parts that we have modified, replaced or taken out. We have identified two key cases for dependency analysis. *First*, change in an entity static structure – e.g. changing a method name from M1 to M2 - typically impacts other system entities "global impact". Existing efforts identify entities that need to be modified – e.g. methods that call method M1. In our approach we support a further analysis by generating signatures of entities to be modified – e.g. as we renamed M1 to M2 we generate another re-aspect with signature specifying locating all method invocation to M1. Thus using the re-aspect locator we pinpoint not only the entities but also the specific lines of code to be updated in every entity. Table 3 shows different examples of system updates along with the further modifications required along with their signatures. Code required for update may be easy to deduce – e.g. changing name from M1 to M2 is easy to handle. But some cases such as adding a new method parameter, requires manual specification by the re-aspect developer to specify how to obtain get/pass the new argument in every method call. *Second,* change in a method body. In this case we use existing techniques of control flow and data flow analysis to make sure that the resultant code is still consistent. Moreover, we compile the resultant code to make sure that no compilation errors have been introduced during the reengineering process and the final binary file is verified using Microsoft PEVerify.

The re-aspects concept is generally extensible. System engineers can specify their own re-aspect type that does actions other than those delivered by our original re-aspect objective – e.g. code-documentation or code-printing-re-aspect is to document, print, or may be translate to another language, code snippets that match a given aspect signature.

Our prototype SMURF delivers model-driven support for re-aspects and applying them onto source code and relevant system models. This helps saving time required to understand the target system and in conducting system maintenance tasks by using a more visual and model-driven approach rather than using re-aspects scripts. SMURF, while re-engineering, takes into consideration updating other system models such as system features, architecture, design, aspects, etc. This helps solving the inconsistency problem, between the system source code and models.

X. SUMMARY

We introduce a new multi-tenancy reengineering approach – SMURF – that help software engineers in migrating their applications to support multi-tenancy. SMURF is based on the re-aspects concepts where software engineers model the modifications they need to apply on their systems in terms of reengineering aspects. A re-aspect capture the signature of system entities to be modified in terms of OCL constraints; actions to be applied including insert, replace, modify, and delete; and code to apply. SMURF then uses these re-aspects' signatures to locate system entities to be modified and applying the actions specified for each re-aspect. We have developed the set of modifications that may be required to migrate a given system to multi-tenancy. We used these modifications' set in validating SMURF against a set of five open source .Net applications. SMURF successfully helped in migrated these applications to support multi-tenancy. We have conducted performance evaluation of SMURF in migrating applications with different sizes. SMURF focuses on updating the original application to support real multi-tenancy instead of using SaaS platforms that wrap the single-tenant applications and simulate multi-tenancy only as security filters.

REFERENCES

[1] Mohamed Almorsy, John Grundy, and Ingo Mueller, "An analysis of the cloud computing security problem," presented at the Asia Pacific Cloud Workshop, Colocated with APSEC2010, Sydney, Australia, 2010.

[2] Peter Mell, and Tim Grance, "The NIST Definition of Cloud Computing," 2009, www.wheresmyserver.nz/storage/media/faqfiles/cloud-def-v15.pdf, (April, 2010).

[3] C. P. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, and A. t Hart, "Enabling multi-tenancy: An industrial experience report," in *Software Maintenance (ICSM), 2010 IEEE Int. Conference on*, 2010, pp. 1-8.

[4] Hong Cai, Ning Wang, Ming Jun Zhou, "A Transparent Approach of Enabling SaaS Multi-tenancy in the Cloud," in *6th World Cong. Services*, 2010, pp. 40-47.

[5] Hong Cai, Ke Zhang, Ming Jun Zhou, et al, "An End-to-End Methodology and Toolkit for Fine Granularity SaaS-ization," in *IEEE Int. Conference on Cloud Computing, 2009*, 2009, pp. 101-108.

[6] Chang Jie Guo, Wei Sun, et al, "A Framework for Native Multi-Tenancy Application Development and Management," in *9th IEEE Int. Conference on E-Commerce Technology and the 4th IEEE Int. Conference on Enterprise Computing, E-Commerce, and E-Services, 2007.* pp. 551-558.

[7] Y. Dunhui, W. Jian, H. Bo, L. Jianxiao, Z. Xiuwei, H. Keqing, and Z. Liang-Jie, "A Practical Architecture of Cloudification of Legacy Applications," in *Services (SERVICES), 2011 IEEE World Congress on*, 2011, pp. 17-24.

[8] T. N. Thomas Kwok, Linh Lam,,, "A Software as a Service with Multi-tenancy Support for an Electronic Contract Management Application," in *Services Computing, 2008. SCC '08. IEEE Int. Conference on*, 2008, pp. 179-186.

[9] I. developerWorks, *Convert your web application to a multi-tenant SaaS solution*. www.ibm.com/develperworks/cloud/library/cl-ultitenantsaas/index.html?ca=drs-

[10] Z. Xuesong, S. Beijun, T. Xucheng, and C. Wei, "From isolated tenancy hosted application to multi-tenancy: Toward a systematic migration method for web application," in *Software Engineering and Service Sciences (ICSESS), 2010 IEEE International Conference on*, 2010, pp. 209-212.

[11] Norihiko Sakamoto, "Construction of Saas-Based e-Learning system in Japan," *FUJITSU Sci. Tech. Journal,* vol. 45, pp. 290-298, 2006.

[12] A. Z. Cor-Paul Bezemer, "Multi-tenant SaaS applications: maintenance dream or nightmare?," in the Joint ERCIM Workshop on Software Evolution and Int. Workshop on Principles of Software Evolution, Antwerp, Belgium, 2010.

[13] Microsoft. (2006, October, 2010). *Multi-Tenant Data Architecture*. Available: http://msdn.microsoft.com/en-us/library/aa479086.aspx