Toward Easy Migration of Client-Server Applications to the Cloud

Jianbo Zheng and Weichang Du
Faculty of Computer Science, University of New Brunswick, Fredericton, Canada

Keywords: Cloud Computing, Multi-Tenancy, SaaS, Client-Server Computing.

Abstract: As an emerging model for the delivery of software services, Software as a Service (SaaS) has come to be a trend in the software industry due to its low investment, flexibility, and accessibility. However, migration of conventional client-server systems and applications to SaaS may involve complicated processes. This paper proposes a framework for helping software developers to migrate conventional client-server applications to SaaS based applications in cloud environments, with multi-tenancy support and without redeveloping or modifying original application software. The migration framework consists of four components: service proxy, data proxy, tenant management, and cloud resources management. The four framework components, together with an original client-server application, can be seamlessly deployed on the cloud as SaaS. The proposed migration framework has been implemented on the Amazon AWS cloud engine.

SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

In recent years, SaaS is rapidly becoming a popular application model for software vendors seeking to reduce IT costs and take advantage of SaaS' inherent flexibility, quick deployment, and scalability. More software companies have embraced cloud computing by planning to or are in process of migrating their software applications or services from customers' local IT environments to cloud environments. However, the migration processes of conventional applications from local infrastructures to cloud environments, running as SaaS applications, turn out to be quite complicated.

Conventionally, many enterprise applications are based on the client-server model. In this computing model, software vendors sell installation packages and licenses to customers and then assist them to deploy the software on customers' own local IT infrastructures. By contrast, in SaaS model, software is deployed on cloud and provides its services via networks. Customers no longer have to purchase or install application software by themselves. Instead, they subscribe and pay for services on demand. In SaaS model, software vendors become service providers and customers become tenants. In SaaS, a single service instance can serve many tenants at same time. Moreover, SaaS supports rapidly onboarding new tenants, which is essential to grow user base of an application. Thus, how to easily migrate a conventional client-server application to cloud has received increasing attention.

Several recent research projects on the issues of migrating conventional applications to the cloud are introduced below.

Gartner (Gartner, 2011) analyzes five ways of migrating existing applications to the cloud: re-host on IaaS, refactor for PaaS, revise for IaaS or PaaS, rebuild on PaaS, and replace with SaaS, and gives advice on how to choose the method.

Chong et al. (Chong and Carraro, 2006) proposes a SaaS maturity model, which takes configurability, multi-tenancy efficiency, and scalability as the key attributes of SaaS and classifies SaaS software as four maturity levels from level 1 (configurability) to level 4 (all the four attributes), where a higher level is distinguished from its immediate lower level by adding one more attribute.

Guo et al. (Guo et al., 2007) proposes a multi-tenancy enabling programming model and framework, consisting of a set of approaches and common services, to support and speed up multi-tenant SaaS application development. Cai et al. (Cai et al., 2010; Cai et al., 2009) proposes an end-to-end methodology and toolkit for transforming existing web applications into multi-tenant SaaS applications. However, using this migration methodology, developers have to modify the original application software as well as server configurations. Song et al. (Song et al., 2011) defines a SaaSify Flow Language

(SFL) and proposes a SFL tool which would help convert Java web applications to SaaS applications.

Furthermore, enabling applications to support multi-tenancy either during application development or by adapting existing web applications to support multi-tenancy has also been investigated in (Mietzner et al., 2008; Wang et al., 2009; Zhang et al., 2009; Chinchani and Iyer, 2004).

In this paper, we propose a migration framework named Application to SaaS Framework (A2SF). The A2SF aims to help software developers or vendors to transform their conventional client-server software applications to multi-tenant SaaS applications without modifying their original software.

The rest of the paper is organized as follows: Section 2 describes the design and implementation of A2SF and discusses several strategies to implement A2SF on the Amazon AWS Cloud platform. Section 3 introduces a case study on the cloud migration of a real-world client-server application using A2SF on AWS. Finally, the concluding remarks and future work are given in Section 4.

2 FRAMEWORK DESIGN AND IMPLEMENTATION

2.1 Overview of A2SF

The objective of this research project is to design and implement a general cloud-based multi-tenancy framework, namely A2SF, which can be used for easy migration of conventional client-server applications to the cloud with none or less modification of original software.

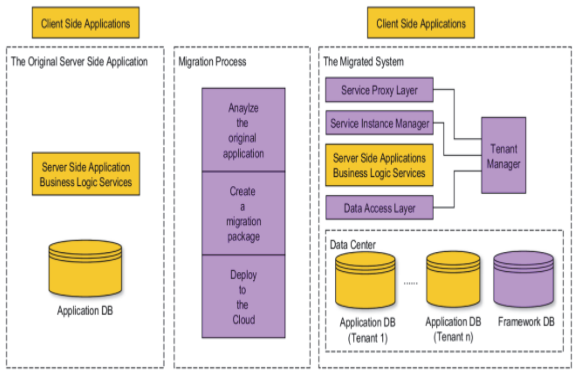

Client Side Applications
Client Side Applications
The Original Server Side Application
Server Side Application Business Logic Services
Application DB
Migration Process
Analyze the original application
Create a migration package
Deploy to the Cloud
The Migrated System
Service Proxy Layer
Service Instance Manager
Server Side Applications Business Logic Services
Data Access Layer
Tenant Manager
Data Center
Application DB (Tenant 1)
......
Application DB (Tenant n)
Framework DB

Figure 1: A2SF overview.

Figure 1 shows an overview of A2SF framework. The purple parts in the figure are the components of A2SF, which provide the backbone of the migrated system, as well as the general migration process.

In Figure 1, the left side is a typical original client-server application, which has three tiers: presentation tier, business logic tier, and database tier. The right side is the migrated SaaS of the original application. The components between the left and right sides are the A2SF migration process. These A2SF components are introduced in details in the following subsections.

2.2 A2SF Challenges and Functionality

There are three challenges of migrating conventional client-server applications to the cloud: tenant management and identification, tenant isolation and data security, and automatic service scalability.

2.2.1 Tenant Management and Identification

Usually, a conventional client-server application has its own client authentication module and customization module, but not tenant management module or even the concept of tenant. This implies that clients of a client-server application usually for a single company or organization. Whereas, a SaaS based application should simultaneously support clients for different companies or organizations, i.e. different tenants.

For this challenge, A2SF provides a general tenant management module to manage the tenant information and configurations. For the tenant identification aspect, the framework assigns a token to each tenant, which will be carried in each service request from clients. Via the token, the tenant identification module is able to identify which tenant the service request is from. This module, working together with the authentication module of the original application, provides identification and authentication functionalities for the migrated SaaS application.

2.2.2 Tenant Isolation and Data Security

As clients from multiple tenants may access the migrated SaaS application at the same time, the security of tenant privacy is an essential requirement to the migrated application. The solution of data isolation and protection is taken as a high priority consideration.

In A2SF, multiple approaches of data isolation are applied. One approach is virtualization-based

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

isolation. By this approach, in the migrated SaaS application, the runtime data and configurations of different tenants are stored in different virtual machines. Another approach is application-based isolation. A2SF provides two types of multi-tenant awareness layers, which execute the access control on services and resources (data) in the migrated SaaS application. In the application level, tenants are only allowed to access their own services and resources.

2.2.3 Automatic Service Scalability

A2SF implements the virtualization-based isolation by dynamically assigning each tenant a virtual machine and deploying the tenant-customized application on the virtual machine. However, if A2SF simply generates a virtual machine for each tenant, the migrated SaaS application could only achieve the first level of the SaaS maturity model (Chong and Carraro, 2006) and would not be able to have high scalability and reduce the operation cost.

Instead, in the migrated SaaS application by A2SF, cloud resources are allocated and recycled in the unit of service instance. In order to have high scalability, A2SF has a service instance manager module, which is responsible for generating and recycling service instances based on the tenant application management rules.

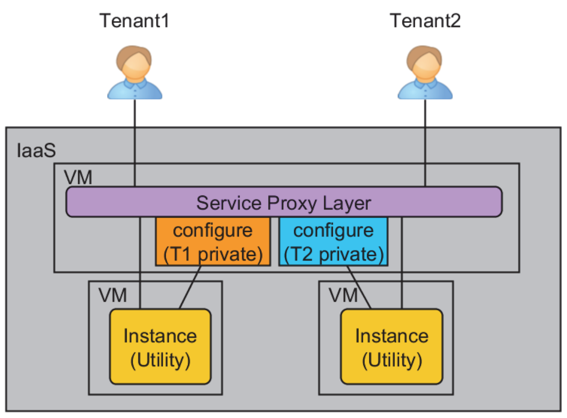


Figure 2: The solution for the application management.

Figure 2 shows the A2SF solution for the application management. In this solution, the original application is divided into two parts. One part is the utility components, which can be shared and reused among tenants. The other part is the tenant privacy related components, for example the configuration component of the application. In A2SF, these tenant privacy related components are protected by the multi-tenant awareness layers.

When the migrated SaaS application needs to generate an application instance for a returning tenant, A2SF loads the tenant's privacy related components to a shared virtual machine instance and assign it to the tenant. When the tenant goes off line, A2SF saves the tenant's privacy related components back to the data centre, clear and recycle the allocated virtual machine instance for future use.

Working like a load balancer between clients and service instances, A2SF helps the migrated SaaS application to allocate and recycle the resources automatically. In this way, the migrated SaaS application can partially achieve the 4th level in the SaaS maturity model without modifying the original application software.

2.3 A2SF Runtime Architecture

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Figure 3 shows the runtime architecture of the A2SF. The purple modules are the main components of the framework, which includes the service proxy layer, tenant manager (including tenant manager interface), service instance manager, data access layer, and framework database.

The service proxy layer and data access layer are the multi-tenant awareness components in A2SF. The tenant manager provides the tenant information management service and tenant identification service. The service instance manager is responsible to automatically generate and recycle service instances based on statuses of real-time tenant access.

The framework database stores the A2SF configuration and tenants' information, such as tenant proxy rules, tenant statuses, and logs.

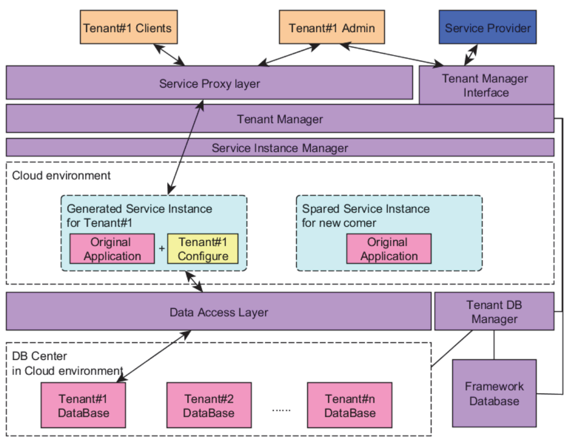


Figure 3: A2SF runtime architecture.

2.3.1 Service Proxy Layer

The service proxy layer is responsible for service access control. In A2SF, tenants are only allowed to access their own service instances. As shown in Figure 3, the service proxy layer is the single entrance of the migrated SaaS application, and all clients or end users' service requests are sent to it, instead of their original local servers.

When the service proxy layer starts up, it loads all the tenants' proxy rules. After receiving a client request, the service proxy layer first acquires the token of the request and identifies the request's source by invoking the tenant identification service. Once the request is identified, the service proxy layer handles the request based on the tenant's proxy rules and forwards the request to the proper service instance. If the request cannot be identified, the service proxy layer will deny the service request.

As the original client-server application will be migrated to the cloud as a whole package without modification, the original communication protocols as well as the business logic processes between clients and server do not need to be changed in the migrated SaaS application as well as the client-side application.

2.3.2 Tenant Manager

The tenant manager provides the following services: tenant information management, tenant identification service, and tenant status service.

Through the tenant manager interface, on one hand, tenant administrators can manage tenants' information, customize tenants' subscribed services, and review tenants' usage reports. On the other hand, the service provider, namely the vendor of the migrated SaaS application, can manage the tenants' subscriptions, set the customization rules for the tenants, and setup the configuration of A2SF runtime.

The tenant identification service is responsible to verify the tenant's token, identify the tenant, and return the tenant information including the tenant rules.

The tenant status service is used to obtain the tenants' current statuses. The tenant manager keeps all the tenants' statuses and updates them regularly, based on the tenants' service requests. In A2SF, the tenant status information includes the tenant service instance status, tenant connection number, tenant live time, and so on. Based on these statuses, the tenant manager sends the service instance management request to the service instance manager, such as service instance generation request and service instance recycling request.

2.3.3 Service Instance Manager

The service instance manager is the module to manage the running service instances in the cloud, including service instance generation and recycling. With this module, A2SF implements the allocation and recycling of the cloud resources.

The service instance is a server side application, which is composed of a running virtual machine instance and the customized application image deployed on the instance. In A2SF, each online or live tenant who has clients or end-users to access the migrated SaaS application is assigned a customized service instance. Whether or not to generate a service instance is controlled by the tenant manager. As discussed in the previous section, the tenant manager keeps the tenants' statuses and updates them regularly. When a tenant status matches its rule of generating or recycling the service instance, the tenant manager will invoke the services provided by the service instance manager, to generate or terminate the service instance for the tenant.

A. Service Instance Generation Steps

When the service instance receives a service instance generation request for a returning tenant, firstly, the service instance manager obtains an available virtual machine from the cloud. Secondly, the service instance manager deploys the original application on the virtual machine. Thirdly, the service instance manager customizes the original application by loading the tenant's private data to the virtual machine from the data centre and initiates the application. Finally, the service instance manager sends the result and service instance information to the tenant manager to update the tenant's statuses.

B. Service Instance Recycling Steps

When the service instance manager receives a service instance recycling request for an off-line tenant, firstly, the service instance manager stops the application. Secondly, the service instance manager collects the tenant's local private data on the virtual machine instance and saves it to the data centre. Finally, the service instance manager clears the tenant's local private data, terminates the virtual machine instance, and sends the result to the tenant manager to updates the tenant's status.

In order to perform the above steps, the service instance manager module wraps the standard operations of the IaaS services (launch, start, stop, and terminate a virtual machine), as well as the remote controlling operations or commands (copy, move, service start, service stop and so on) on the virtual machine.

Moreover, the virtual instance launching time may be too long for an application that needs quick responding. So in order to shorten the generation time of service instances, A2SF also implements a virtual pool managed by the service instance manager, to always keep a number of spare virtual machines that are waiting to host generated service instances.

2.3.4 Data Access Layer

The data access layer is responsible for the data access control between service instances and the database, to assure that tenants are only allowed to access their own data.

Generally speaking, an application has two essential parts, programming code and user data. The programming code is the common part and can be shared with all tenants, while the user data are tenants' private data which are protected under the access control. Furthermore, the user data can be categorized into local data and remote data. The local data are stored in local files. For instance, the configuration file is a typical example of local data. The remote data is usually stored in a database.

The data access layer stores the local data in the A2SF's data storage (such as S3 in the Amazon Cloud), initializes local data before the service instance runs, and stores and clears it after the service instance stops.

The data access layer provides two ways to deal with the remote data. One way is to implement a data proxy server, which means that all data accesses go through the data proxy server first. This method is for those applications whose data connection configurations are hard coded. The other way is to treat the data connection configuration file as tenant's private local data. This method is only suitable for those data connection configurations that are separated from the code and can be easily replaced.

2.4 Migration Process

Generally, the process to migrate a client-server application to SaaS using A2SF consists of three steps: analyze the original application, create a migration package, and deploy the package to the cloud.

2.4.1 Analyze the Original Application

Before starting to integrate the A2SF components and the original application, the software vendor needs to analyze the original application to identify the tenant privacy related components of the original application and separate them from utility or sharable components.

2.4.2 Create a Migration Package

Based on the analysis of the original application, a migration package can be created to integrate the A2SF components and the original application software together to be ready for deployment to the cloud.

The migration package consists of A2SF components of service proxy server, tenant manager, service instance manager, data proxy server, as well as A2SF script templates, plus the original application software.

An A2SF script is an executable batch file to be created for each tenant for the tenant's application customization and data protection. There are two types of script templates. An initializing script will be executed in the application generation. When an application instance needs to be generated, the initializing script will install the application first, then customize the application, copy the tenant's private data files from the A2SF data centre to this application instance, and launch the application instance. A context-saving script is to be executed during the application recycling. After the tenant goes off line, the context-saving script will stop the tenant's application instance, save the tenant's current context, copy the tenant's private data files to the data centre, and clear the tenant private data in the application instance. Using the script templates, A2SF runtime can automatically generate the concrete scripts for each subscribed tenant, according information provided by the tenant during subscription time.

2.5 A2SF Implementation

A prototype of A2SF has been implemented on Amazon AWS cloud platform. Figure 4 shows the organization of the implementation. Three Amazon cloud services were applied to the prototype: EC2 service, S3 service, and RDS service. All the framework components were running on the Amazon Linux operation system.

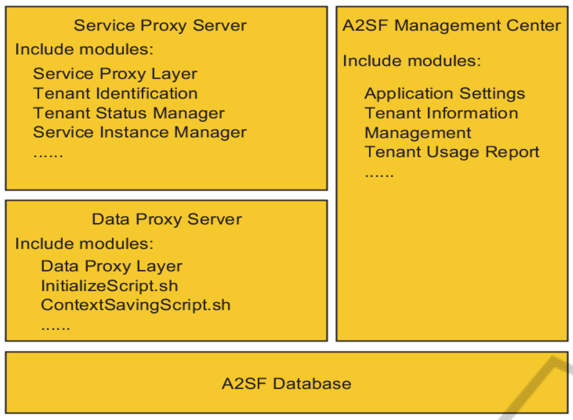


Figure 4: Main components of the A2SF prototype.

3 CASE STUDY

In this section, a case study of migration of a real-world client-server application to SaaS on Amazon cloud using A2SF is presented, to show the migration process and the migrated SaaS services with the A2SF prototype.

3.1 SugarCRM Application

In order to find a typical and representative real-world client-server application for this case study, we searched for a popular open source CRM application. SugarCRM is one of the most popular customer relationship management applications currently. It is implemented in the PHP programming language and supports multiple types of databases including MySQL.

3.2 Migration of SugarCRM to SaaS

3.2.1 Analyze Application

We analyze the SugarCRM application from the following three aspects.

A. Separate Tenant Privacy Related Components

As discussed in previous sections, the components of the original client-server application should be classified into three types: utility components, local private components, and remote private components. We need to identify the tenant's privacy related components in SugarCRM software and separate them from the utility components.

A2SF provides different migration solutions for the different types of components. Utility components are deployed on the virtual machine image (in the Amazon cloud, it is also called as AMI) once for all the tenants. The local private components are saved in the bucket of Amazon S3 named by tenant id and tenant name. The remote private components are stored in a MySQL instance in the Amazon RDS.

After installing SugarCRM on an experiment environment, we classified the components of SugarCRM as shown in Table 1:

Table 1: The classification of SugarCRM components.

Type	Components
Local private components	Folders: custom, upload, cache, data, modules; Files: .htaccess, sugarcrm.log, config.php, config_override.php
Remote private components	The database "sugarcrm"
Utility components	The rest of components of SugarCRM

B. Determine the Types of Cloud Services

For this case study, the basic types of cloud services are enough for the migrated SaaS application. For example, from the Amazon cloud services, we chose the micro type virtual machine instance for both the service proxy server and service instance server. We chose the micro type database instance and the basic S3 storage as the data centre.

3.2.2 Create a Migration Package

In the next step, we integrated the A2SF components and the SugarCRM software to create a migration package.

Firstly, we updated the A2SF script templates by our Amazon Security Credentials and the list of private components.

Secondly, a customized AMI (Amazon Machine Image) was created for generating SugarCRM service instances, in which the data proxy server, the A2SF scripts, the target application SugarCRM, and the running environments were already deployed. The AMI ID of the A2SF was updated by the A2SF management centre. Thus when a new or returned tenant comes, the service proxy server can easily and quickly launch a new SugarCRM service instance and customized it for the new tenant based on the AMI. Table 2 shows the contents of the created migration package for SugarCRM to SaaS on Amazon cloud.

Table 2: Contents of SugarCRM migration package.

Component	Description
Service proxy server	Java jar file to run on the portal server VM as the service entrance.
Management centre	Java war file to be deployed to Tomcat server running on the portal server VM
Data proxy server	Java jar file to be pre-deployed in the AMI, and run as the local MySQL proxy server VM.
Script template	Bash script file to be instantiated by tenant subscriptions
Customized AMI	EC2 VM machine image file with pre-deployed data proxy server image
Database	Pre-defined MySQL database for A2SF framework.
SugarCRM Application	Original client-server application
SugarCRM DB Creation Script	SugarCRM's database creation script file to be executed when a new tenant's subscription is approved.

3.2.3 Deploy the Package

In this stage, the migration package was deployed and ran on the Amazon cloud. Figure 5 shows the deployment of the migrated SaaS SugarCRM application.

Firstly, we created a new virtual machine from Amazon EC2 as the portal server and a database named "a2sf" in the MySQL instance of Amazon RDS. The portal server is the entrance of the migrated system, as well as the host of the service proxy server and the A2SF management centre.

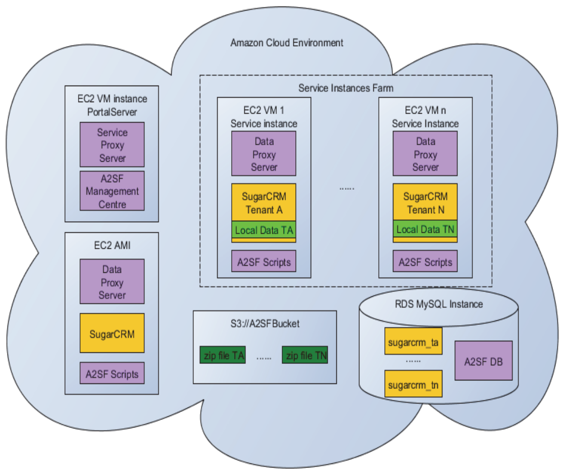
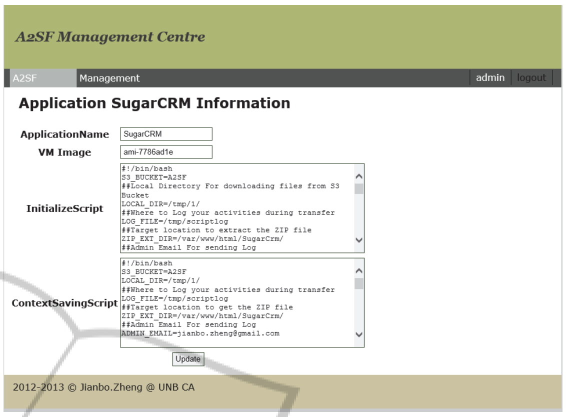


Figure 5: Deployment of the migrated SugarCRM SaaS.

After the portal server running, we accessed the A2SF management centre and updated the application information. As shown in Figure 6, currently, the application information includes the application name, the virtual machine image id (AMI ID), and the A2SF scripts.



Application SugarCRM Information

ApplicationName: SugarCRM
VM Image: ami-7786ad1e

InitializeScript:

```
#!/bin/bash
S3_BUCKET=A2SF
##Local Directory For downloading files from S3
Bucket
LOCAL_DIR=/tmp/1/
##Where to Log your activities during transfer
LOG_FILE=/tmp/scriptlog
##Target location to extract the ZIP file
ZIP_EXT_DIR=/var/www/html/SugarCrm/
##Admin Email For sending Log
```

ContextSavingScript:

```
#!/bin/bash
S3_BUCKET=A2SF
LOCAL_DIR=/tmp/1/
##Where to Log your activities during transfer
LOG_FILE=/tmp/scriptlog
##Target location to get the ZIP file
ZIP_EXT_DIR=/var/www/html/SugarCrm/
##Admin Email For sending Log
ADMIN_EMAIL=jianbo.zheng@gmail.com
```

2012-2013 © Jianbo.Zheng @ UNB CA

Figure 6: Screenshot of updating application's information.

After starting up all the components in the cloud, the clients or end users of different tenants can access the migrated SugarCRM SaaS from their web browsers on different client computers, where we use client computers' IP addresses as tenants' tokens to identify which tenants the clients belong to.

4 CONCLUDING REMARKS

In this paper we proposed a framework named A2SF for migrating conventional client-server applications to multi-tenant SaaS applications. Migrations based on A2SF are relatively easy because no need to modify the original software and the simple migration process. The runtime architecture of A2SF for multi-tenant SaaS is simple and effective, though it is little conservative compared to other proposed multi-tenant SaaS systems. The proposed A2SF framework has been implemented on Amazon EC2 cloud computing engine with a real-world migrated CRM application. Future work includes performance evaluation and improvement of A2SF and implementations on other cloud platforms.

REFERENCES

Cai, H., Wang, N., & Zhou, M. J. (2010, July). *A transparent approach of enabling SaaS multi-tenancy in the cloud.* In Services (services-1), 2010 6th world congress on (pp. 40-47). IEEE.

Cai, H., Zhang, K., Zhou, M. J., Gong, W., Cai, J. J., & Mao, X. (2009, September). *An end-to-end*

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

methodology and toolkit for fine granularity SaaS-ization. In Cloud Computing, 2009. CLOUD'09. IEEE International Conference on (pp. 101-108). IEEE.

Chinchani, R., & Iyer, A. (2004). HNQ, and S. Upadhyaya. *A Target-Centric Formal Model for Insider Threat and More.* Technical Report 2004-16, University of Buffalo, US.

Chong, F., Carraro, G. (2006). *Architecture strategies for catching the long tail.* MSDN Library, Microsoft Corporation, 9-10.

Gartner, "*Gartner Identifies Five Ways to Migrate Applications to the Cloud*" 16 May 2011. [Online]. Available:http://www.gartner.com/it/page.jsp?id=1684114

Guo, C. J., Sun, W., Huang, Y., Wang, Z. H., & Gao, B. (2007, July). *A framework for native multi-tenancy application development and management.* In E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on (pp. 551-558). IEEE.

Mietzner, R., Leymann, F., & Papazoglou, M. P. (2008, June). *Defining composite configurable SaaS application packages using SCA, variability descriptors and multi-tenancy patterns.* In Internet and Web Applications and Services, 2008. ICIW'08. Third International Conference on (pp. 156-161). IEEE.

Song, J., Han, F., Yan, Z., Liu, G., & Zhu, Z. (2011, July). *A SaaSify tool for converting traditional web-based applications to SaaS application.* In Cloud Computing (CLOUD), 2011 IEEE International Conference on (pp. 396-403). IEEE.

Wang, D., Zhang, Y., Zhang, B., & Liu, Y. (2009, December). *Research and implementation of a new SaaS service execution mechanism with multi-tenancy support.* In Information science and engineering (icise), 2009 1st international conference on (pp. 336-339). IEEE.

Zhang, X., Shen, B., Tang, X., & Chen, W. (2010, July). *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 (pp. 209-212). IEEE.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS