Software Customization Based on Model-Driven Architecture Over SaaS Platforms

Xiyong Zhu
Department of Management Sciences and Engineering
Zhejiang Sci-Tech University
Hangzhou, China
zxywolf@163.com

Shixiong Wang
Department of Management Sciences and Engineering
Zhejiang Sci-Tech University
Hangzhou, China
xim_wang@163.com

***Abstract*—Software-as-a-Service (SaaS) is changing the way enterprises develop and deploy information systems. With the rapid development and successful implementations of the model-driven architecure(MDA) and service-oriented architecture (SOA), SaaS has become the focus of research. This paper describes the design and implementation of a service template markup language (STML) and its integrated development tools, which provide a comprehensive solution to the customization of applications over SaaS platforms for individual users on an as-needed basis. In the paper, we first analyze and point out the limitations of current software customization approaches in SaaS platforms, then propose a framework for a MDA-based approach to software customization in a SaaS platform, next present a series of SaaS service models and an XML-based service template markup language, finally conclude by an example showing the practicability of the STML tools.**

Keywords- SaaS, MDA, software customization, STML

I. INTRODUCTION

In recent years, with the rapid development of the Internet technologies and the widespread acceptance of web-based applications in a variety of enterprise systems, Software-as-a-Service (SaaS) has become the focus of research. SaaS offers software products, such as Enterprise Resource Planning (ERP) systems and Customer Relationship Management Systems (CRM), to multiple users as services available through networks such as the Internet. Users could pay only a monthly rental charge or a subscription fee to SaaS vendors to use these software systems. Many software vendors (e.g. IBM, Oracle, SAP, Peoplesoft, Salesfore, etc.) have already built their SaaS platforms and put their commercial software on theses platforms for users to rent from the Internet. These SaaS platforms have great appealing not only to small enterprises with little or no IT staff, but also to medium-large enterprises with established IT departments. Predictably, the association of commercial software products with SaaS will become the dominant trend for future software development and deployment.

However, software must be technically general-purpose and customizable before they can be leased over a SaaS platforms [1]. Traditional application service provider (ASP) (i.e. appeared in the late 1990s) emphasized only the lease services for software products, and most leased software systems were only those highly standardized applications such as accounting and inventory management. These systems had normalized business processes and pre-determined programming structures, thus seemed rather inflexible to user's specific requirements [2]. SaaS, however, aimed to providing complete solutions to an enterprise's informationalization, emphasize the flexibility and personalization of the software products for different users, because the leased software systems have to cover almost all aspects of an enterprise business management, including many complex or semi-structured business processes. Many of these systems cannot be pre-programmed and must be customized in SaaS platforms as to meet user's specific requirements on an on-demand basis.

This paper describes the design and implementation of a SaaS platform, named STML. It provides a comprehensive solution to the development of customizable applications over the SaaS platforms, in which technology and computing resources are made available to its individual users on an as-needed basis.

II. VERSION-BASED APPROACHES TO SOFTWARE CUSTOMIZATION AND IT'S SAAS ARCHITECTURES

In a version-based approach[3], the SaaS platform hosts a single instance of the application for each user, all instances use the same code implementation, and the vendor meets users' needs by providing detailed configuration options that allow the user to change how the application looks and behaves to its users. Authorization and security policies ensure that each user's data is kept separate from that of other customers.

The main strategy for version-based approach is to develop a full-functional, general-purpose and highly standardized software product, and pre-defines some parameters on certain functional modules that allow users to describe the software by setting these parameters [3]. Customization of version-based software systems may involve in altering the input or output layouts, configuring the data sources, and redirecting of some business processes. The ways for customization of these software systems are always straightforward and are finished simply by means of menus selections, parameter configurations, and certain visual designing tools, because all are limited within the pre-determined scope on the version basis. Generally, the version-based software products always have certain independent configuration modules or tools (often enclosed in software's system initialization modules)[4]. Some popular commercial software vendors (including some ERP

978-1-4244-4639-1/09/$25.00 ©2009 IEEE

systems) adopt this approach to implement limited customization of their products.

However, to lease version-based software systems in a SaaS platform for multiple users, it is necessary to store each user's configuration data in separate places. When the SaaS platform is running an application for a user, it has to first get access to his configuration values, then set these values to the related parameters, and continue to run the system according to these settings. Obviously, the SaaS architecture based on this approach has many limitations, including: (i) the subjects and aspects of businesses for customization must be pre-determined and the scalability of the application is limited. It is only suitable to lease simple, general-purpose, and highly standardized software systems, whose structures and procedures are always pre-written and inflexible. Moreover, software customization based on this architecture do not really tailor or minimize the system resources to the exact needs from user's specific requirements; (ii) different users are running on the same software system in a SaaS server, the server could be heavily overloaded and exhausted when delivering software over the Internet if hundreds of end-users are running on the platform same time; (iii)different kinds of software configuration data are not stored and presented in a unified or normalized form, making it difficult for both developers and end-users to maintain, read or trace these settings.

III. A MODEL-DRIVEN APPRAOCH OF SERVICE-ORIENTED ARCHITECTURE FOR SAAS PLATFORMS

A. Overall architecture

Model-driven architecture (MDA) is a kind of domain engineering software design approach for the development of software systems. The MDA approach defines system functionality using a platform-independent model (PIM) using an appropriate domain-specific language. Then, given a platform definition model (PDM), the PIM is translated to one or more platform-specific models (PSMs) that computers can run. The PSM may use different Domain Specific Languages, or a General Purpose Language like Java, C#, PHP, etc. Automated tools generally perform this translation[5].

A service-oriented architecture (SOA) is a distributed software model, whose goal is to achieve loose coupling among interacting software components or agents. A SOA is usually comprised of three primary parties: a provider of services, a consumer of services, and a directory of services. A service is a unit of work done by a service provider to achieve desired end results for a service consumer. In a SOA, services are used to divide larger applications into smaller discrete modules. At same time, services are integrated via service composition mechanisms to create larger applications[6][7].

This paper proposes a MDA-based of SOA approach to software customization over the SaaS platform by introducing the cconcept of service models (or templates) and a domain-specific language of STML (Service Template Markup Language). The main modules of the STML tools are the Web Portal, the Service Customization and Integration Tools (SCIT), the Service Execution Platform (SEP), and the Service Templates Repository (STR). The STML Portal provides users with the necessary interface in order to access applications on-demand services. It is also responsible for the administration (including registration, authorization, etc.) of users. The SCIT provide tools for users to edit, check, verify, and store service description files in STML language, and translate these files into source code programs. It is also responsible for the composition and integration of multiple services. The SEP invokes and executes user's on-demand applications via the Internet. It is also responsible for downloading the appropriate parts of the application according to the user's actions, while enforcing the mutually agreed frame between the user and the application service provider. The STR is a collection of semi-programmed applications that can be transformed into executable programs by the Automatic Code Generator in the Service Customization Tools. So far, the STML tools generate both ExtJS-based Java and Delphi source code programs. Fig. 1 is an illustration of the STML platform architecture.

One important feature of this architecture is that service is a higher level of abstraction for business processes. In a service, data and procedures are embedded into an application unit. By using the concept of services, software customization can be simplified into service customization, which significantly decreases the implementation complexities.

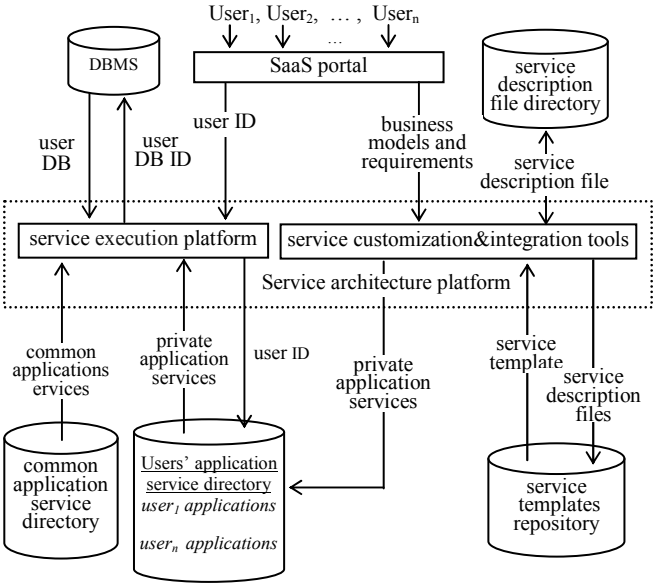

FIG. 1 SAAS ARCHITECTURE FOR MDA-BASED APPLICATION CUSTOMIZATION

B. SaaS service templates repository

The idea of template is derived from the concept of objects or components, but template differs greatly in construction and execution from component. Template is a nearly complete application model where the completed parts include an application's architecture and reusable components. The architecture is generic to fit any application in the class, and the reusable components are designed to work with each other within the architecture.

The STML service template is defined as a technically typical and functionally general-purpose module or subsystem

that is abstracted or derived from various e-business systems. It represents those business services that are common to a very large set of business domains. A service template serves at least one specific or even very comprehensive business function, which is general-purpose, flexible, and can be easily combined to build business applications.

A service template has both common and variant parts of attributes. The common part consists of data and procedures that can be pre-defined, pre-designed, pre-programmed, pre-documented and stored, such as function boundaries, programming flowcharts, input and output layouts, related database or tables, semi-complete source code programs, and so forth; The variant part consists of parameters and procedures that need to be specified later because they are not general-purpose and cannot be pre-determined.

C. *MDA-based service description*

In the STML, business models are decomposed into services and are described in the STML language. The STML is XML-based, most of its tags and labels are common to all types of service descriptions, but some are exclusively oriented to specific templates. So far, the STML is able to specify the following types of specifications within a service template: (i) type of the service; (ii) data sources, including the DBMS, the databases, and the tables; (iii) service contents, including the inputs, outputs, and data processings; data validation rules, reports, etc.; (iv) messages. Table I is a list of some common STML tags, usages and descriptions.

TABLE I COMMON STML TAGS AND DESCRIPTIONS

Tag name	Description
<service type=<*servicetype*>⋯ ⋯ </service>	Specify the beginning of the service description and the service type
<inputs type="<controltype>" [page="<pageid>"] [panel="<panelid>"] [name="<fieldname>"] [pos="<y1,x1>" [size="<y2,x2>"] [fontsize="<fontsize>"] [fontname="<fontname>"] [fontstyle="<b/u/i/n>"] [datafields="<fieldslist>"] [table="<tablename>"] [masterfields="<fieldslist>"] [items="<itemslist>"] [indexfields="<fieldslist>"] [filter="<fieldexpression>"] [servicefile="<STML filename>"] />	Add a control (or a component) into the form, and specify it's properties. So far, the SaaS- STML supports the more than 100 control types, typical examples are: page, panel, label, text, labeltext, title, keyfield, dbgrid(mastergrid, detailgrid), treeview, combobox, memo, imagefile, image, editbutton, grideditbutton, gridcombobox, menubar, summery, grandtotal, sysnumber, sysdate, sysmaker, syschecker, etc. Each type of control has it's own set of tags to describe it's input, output, or processings.
<table| mastertable| detailtable id="<tableid>" | name="<tablename>" />	Specify either the table name or the table identity within a database
<validation>logic expression </validation>	Specify data input validation rules
<replace>expression </replace>	Specify an expression for a computed field in a table
<totalfields>fieldlists </totalfields>	Specify the fields to be summarized in a table
<sqlstring>sql select statement </sqlstring>	Specify the SQL statement in a service for printing reports
<subservice name="<STML filename>" value="<button caption>" action="<actionlists>" />	Specify the filename of a subservice and when it is to be invoked.

D. *Service integrations*

In the STML, service composition and integration are implemented mainly by two approaches of menu-driven and service nesting. As menus are dynamically created in the SaaS platform, different services can be integrated into a whole software system by the menu-driven integration.

Service nesting means that a master service description file can contain STML tags (e.g. <subservice>, as listed in table I) to specify another service file (called subservice) in it. The SaaS platform can automatically find the URLs or call the corresponding application programs when a subservice is invoked.

IV. EXAMPLE OF SOFTWARE CUSTOMIZATION IN THE STML-BASED SAAS PLATFORM

A. *A service template for accounting documents management*

The following example illustrates how we use the STML language and the service customization tools for rapid development of applications in the SaaS platform.

In e-business systems (typically in ERP systems), there are a number of business processes involving the use of an application service to create and maintain various kinds of documents, such as orders, receipts, bills, vouchers, invoices, etc. Figure 2 is the illustration of a service template, which is originally designed for accounting documents management, but can be also used to treat all above-mentioned and other documents.

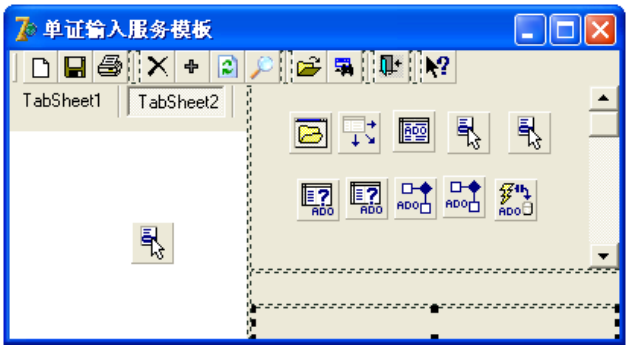

FIG 2 AN APPLICATION SERVICE TEMPLATE CREATED ORIGINALLY FOR ACCOUNTING DOCUMENTS MANAGEMENT

In the service template above, there are some pre-determined components, such as a toolbar, a treeview, a pagecontrol, a dbgrid, and many other controls which are common to all those applications based on the template. User's specific requirements including the data sources, the inputs, the validation rules, the messages, and other configurations are defined in a service template description file by using the STML language.

B. *Service description by using the STML*

By using the STML, we create a service description file specifically for orders management application. Table II is a STML file used to specify the configurations for that application.

Fig. 3 shows the presentation and screen layouts of the order management application when it is running on the SaaS platform or on user's local machines, corresponding to what is defined in its STML file. The application has a variety of functions, including adding, updating, deleting, searching, and printing records into or from the orders table.

TABLE II SERVICE DESCRIPTION FOR ORDERS MANAGEMENT

```
<?xml version="1.0" encoding="GB2312" ?>
<service type="document_input_a">
  <title>Order Input</title>
  <table name="x_orders" masterfields="sysbillid" />
  <input type="treeview" datafields="sysdate;orderid" size="0,30" />
  <input type="title" value="Orders Input" pos="1,40" fontsize="16"
    fontname="times new roman" />
  <input type="sysnumber" value="Order No:" pos="3,2" />
  <input type="labeltext" name="orderid" value="Order ID:" pos="5,2"
    size="1,16" />
  <input type="sysdate" value="Order Date:" pos="7,2" size="1,18" />
  <input type="editbutton" name="customerid" value="CustomerID:"
    pos="9,2" size="0,16" datafields="[14]customerid/customerID; [40]
    customername/ CompanyName"  table="x_customers/Customers" />
  <input type="labeltext" name="customername" value="CompanyName:"
    pos="11,2" size="0,46" style="readonly" />
  <input type="datetime" name="RequiredDate" value="Required Date:"
    pos="13,2" size="1,18" />
  <input type="labeltext" name="Freight" value="Freight:" pos="13,40"
    size="1,12" />
  <input type="combobox" name="employeename" value="Employee:"
    pos="16,2" size="0,20" datafields="employeename"
    table="x_employees/employees" />
  <input type="sysmaker" value="Operator:" pos="16,40" />
  <validation>customerid!=''</validation>
  <validation>orderid!=''</validation>
  <detailtable name="x_orderdetails" value="Order Details"
    masterfields="sysbillid" />
  <input type="detaildbgrid" datafields="[c10]productid/ProductID;
    [c18][readonly]productname/ ProductName; [c12][readonly]spec/
    QuantityPerUnit;[N8]qty/Quantity; [n8,2]price/Unitprice;
    [n9,2][readonly]amount/Amount" pos="0,0" size="6,0" />
  <input type="grideditbutton" name="productid" datafields=
    "[15]productid/ProductID;[30]productname/ProductName"
    table="x_products/Products" />
  <validation>productid!=''</validation>
  <input type="summery" name="qty" value="Qty Total:" pos="1,30"
    size="0,12" />
  <input type="summery" name="amount" value="Amt Total:" pos="1,60"
    size="0,12" />
  <replace>[n]amount=qty*price</replace>
  <validation>productid!=''</validation>
</service>
```

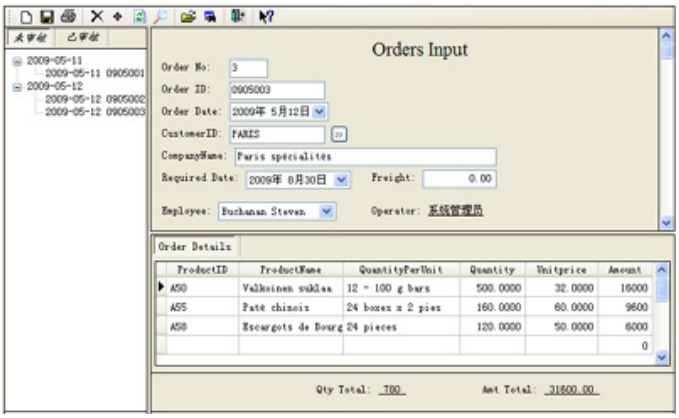

FIG 3 THE ORDERS MANAGEMENT APPLICATION RUNNING ON THE STML-BASED SAAS PLATFORM

V. CONCLUSION

In this paper we presented the design and implementation of the STML language and its integrated tools, proposed a MDA-based of SOA approach to software customization over the SaaS platform. By implementation the MDA and SOA architecture, we presented an XML-based service template markup language to describe the business requirements. This approach not only greatly improves the efficiency of software development, but also improves the latitude and flexibility of the software systems. It reduces the complex development of an on-demand application to a simple definition of the services.

The STML-based SaaS platform is now in the pilot stage with its actual deployment at ten different Garment Import & Export companies for on-demand CRM implementation. We plan to gather the observations and results that will be produced through the usage of the STML tools by other types of enterprises and then possibly use them to enhance the current implementation.

ACKNOWLEDGMENT

This research is supported by the National Natural Science Foundation of China under Grant 70572071.

REFERENCES

[1] Yan XiuQi, Ma DengZhe, Fan FeiYa.(2005). Research and Development on ASP Platforms Oriented to Small-medium Enterprises. *Computer integrated manufacturing system*s of China, (June). 56-59.

[2] Michael Alan Smith , Ram L. Kumar.(2004). A theory of application service provider (ASP) use from a client perspective. *Information & Management*, vol. 41, 977–1002.

[3] Xue Hanxin, DuRao, Huang HuiJun.(2003). Research on Version-based customizable ERP Systems. *HeFei Industrial University Press (Natural Science Version)*, (October). 875-878.

[4] Qi GuoNing, Gu XinJian, Tan JianRong.(2003). *Mass Customization Technologies and Applications*, Publishing House of Engineering Industry, Beijing, China.

[5] Pablo Amaya, Carlos Gonzalez, Juan M. Murillo.(2006). Towards a Subject-Oriented Model-Driven Framework. *Electronic Notes in Theoretical Computer Science,* Vol. 163,31–44.

[6] A. Solberg, D. Simmonds, R. Reddy, S. Ghosh, R. France.(2005). Using Aspect Oriented Technologies to Support Separation of Concerns in Model Driven Development. *Proceeding of the 29th Annual International Computer Software and Applications Conference*, Edinburgh, Scotland, (July). 226-235.

[7] Ying Huang.[2003]. A template-based Approach for service-Oriented e-business Solutions. *Proceedings of the Third International conference on electronic commerce engineering(IceCe2003)*. International academic Publishers, World Publishing Corporation, 670-675.