Software as a Service: An Integration Perspective

Wei Sun[1], Kuo Zhang[1], Shyh-Kwei Chen[2], Xin Zhang[1], and Haiqi Liang[1]

[1] IBM China Research Lab
[2]IBM T.J Watson Research Lab
{weisun, zhangkuo, zxin, lianghq}@cn.ibm.com, {skchen}@us.ibm.com

Abstract. Software as a Service (SaaS) is gaining momentum in recent years with more and more successful adoptions. Though SaaS is delivered over Internet and charged on per-use basis, it is software application in essence. SaaS contains business data and logics which are usually required to integrate with other applications deployed by a SaaS subscriber. This makes Integration become one of the common requirements in most SaaS adoptions. In this paper, we analyze the key functional and non-functional SaaS integration requirements from an industry practitioner point of view; and summarize the SaaS integration patterns and existing offerings; then point out the gaps from both technology and tooling perspectives; finally we introduce a SaaS integration framework to address those gaps. Considering there is no much academic work on SaaS service modeling, we come up with a SaaS service description framework as an extension of Web Service description, so as to model SaaS unique features in a unified way. With the supported tooling and runtime platform, the framework can facilitate the SaaS integration lifecycle in a model-driven approach.

1 Introduction

Software as a Service (SaaS) is a software delivery model, which provides customers access to business functionality remotely (usually over the internet) as a service [1, 2]. The customer does not specially purchase a software license. The cost of the infrastructure, the right to use the software, and all hosting, maintenance and support services are all bundled into a single monthly or per-use charging. As SaaS brings lower Total Cost of Ownership (TCO) and better Return On Investment (ROI), SaaS services achieve a prosperous development and cover most of the well-known application areas, e.g. Customer Relationship Management(CRM) service from Salesforce.com, Human Resource Management(HRM) service from Employease.com [3, 4].

The functionalities of services delivered through SaaS may vary. Complete or full-blown solutions can be costy and hard to configure. Simple services normally provide specific functionalities, but a need exists to integrate several services together to achieve a desired business operation [7]. There are no standards or guidelines for clients to make technical decision. It becomes a critical problem when disparate services come from different software providers, through different service protocols and with various functionalities. The integration requirement of SaaS customers has been studied by different SaaS market research efforts. According to the survey of 639 companies by AMR research, more than 70% companies expect that the SaaS solution can be integrated with their on-Premises legacy applications or other SaaS solutions

B. Krämer, K.-J. Lin, and P. Narasimhan (Eds.): ICSOC 2007, LNCS 4749, pp. 558–569, 2007.

they subscribed/plan to subscribe [8]. IDC conducted a SaaS solution adoption trend study in 2004 and found that more than 50% of the survey respondents selected "Better integration with in-house applications" as one of the top 3 drivers, making SaaS solution more attractive [9].

There are many industry players and offerings addressing the SaaS integration issues. AppExchange from Salesforce.com provides a hosting platform and web based programming tools for third party vendors to develop/integrate add-on SaaS services on top of its CRM service [4]. Jamcracker enables a hosted SaaS integration hub [10]. IBM SaaS showcase provides a portal of different SaaS services which can be subscribed and integrated [11]. OpenKapow focuses on wrapping SaaS services' capability with Web user interface into a standardized component called "Robot", and leverages Mashup technologies to facilitate the composition of "Robot" come from different SaaS services and Web Services [29]. However, there are few academic works in this area. Seltsikas explored the integration challenges for application service providers from business model point of view [12]. Elfatatry studied SaaS from contract negotiation point of view [13]. Turne summarized SaaS related Web Services technologies [14]. Ottinger from Mule open source group [30] analyzed some SaaS integration requirements, which highlighted several SaaS integration unique issues around corporate firewall, network latency of the integration. However there is not a relatively complete and deep analysis about the SaaS integration requirements as well as the demand for any new technologies in a holistic view.

In this paper, we analyze and identify the SaaS integration requirements and technology gaps, and then propose reference architecture of the SaaS integration framework, which includes a SaaS Description Language (SaaS-DL) to support model driven integration approach, tooling and runtime components as well as different configurations. SaaS can cover very broad areas of Web Services, in this paper, SaaS specially focuses on those business applications (e.g. CRM [4, 16]) delivered in Web Services model. The rest of the paper is organized as follows. Firstly, we will analyze the SaaS integration requirements, common patterns as well as challenges in section 2; based on these analysis, a SaaS-DL and integration framework reference architecture will be introduced in Section 3; then in section 4 prototype implementation of the framework is presented; a case study is introduced in Section 5 to illustrate how the integration framework works; finally, conclusions and future work will be summarized in section 6.

2 SaaS Integration Requirements and Patterns

Most SaaS service subscribers, especially those medium to large companies, have certain applications already deployed on their premises. This makes the application environments of those companies become a hybrid model illustrated in the figure 1. SaaS service is usually a web application which can be accessed by different customers through Internet. Just like normal web based business application, SaaS application is composed by three major layers: user interface, business logic, and data. On the other hand, the SaaS application is special. It usually involves the metering and billing for the usage of the service consumer. Its Quality of Service (QoS) should achieve Service Level Agreement (SLA) between the service provider and consumer according to the service contract. In this section, we will explore the SaaS integration requirements from both functional and non-functional perspectives.

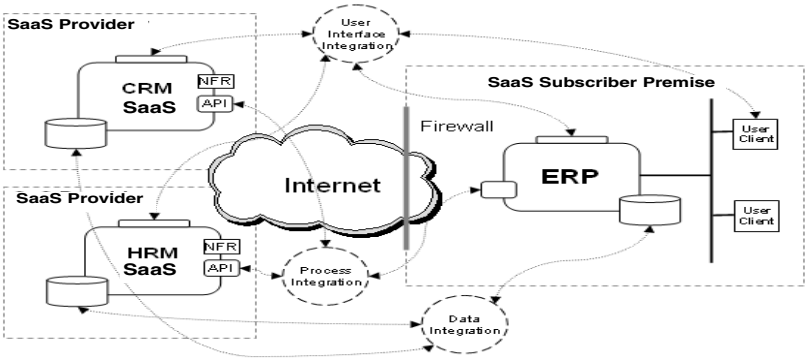

Fig. 1. SaaS Consumption Environment and Integration Requirements

2.1 SaaS Integration Functional Requirements and Patterns

SaaS subscriber leverages SaaS services to support certain business functions, e.g. CRM, HRM. However any business function cannot be isolated from others in most cases. For example, the sales person's commission calculation in HRM should be supported by the sales' performance data managed in CRM. Therefore the different applications/services a company deployed/subscribed should be integrated together. The integration will happen in all the three layers of the SaaS application.

a) User Interface(UI) Integration

Every application has its own user interface and related access control. So the SaaS subscribers should switch among different user interfaces with different user identity and password information required by SaaS services and on-premise applications. As illustrated in figure 2, pattern U-I, Single Sign On (SSO), is a very common UI integration requirement. SSO can enable users log on once and then access all the authorized user interfaces from different applications/SaaS services. Pattern U-II, Mash-up [7, 15], can enable users to access one application/SaaS service's data through another SaaS service/application's user interface.

b) Process Integration

A business process supported by a SaaS service usually can trigger business process supported by another SaaS service or on-premise application. For example, an order process from CRM service should trigger an order fulfillment process managed by ERP application. Therefore process integration can automate the end to end business process transaction span across multiple SaaS services and on-premise applications. There are four key process integration patterns that are usually required. Pattern P-I and P-III can support invoking another process or receiving an invocation through Web Services technology. P-II can support scheduled process invocation in pulling mode. Pattern P-IV can support complex process integration scenario using workflow, in which different people and applications will be involved to link different processes.

c) Data Integration

There are two types of data in a SaaS service: master data and transactional data. As illustrated in Pattern D-I and D-II, these data should be synchronized or migrated from

SaaS services to on-premise applications or vice versa. One type of data in a company's application environment should have only one master data source. The master data source should populate or synchronize the data to other applications/SaaS services timely that need to store that data locally. For example, if a company subscribed a CRM SaaS service, the customer information related data should be a type of master data maintained by CRM SaaS service, though ERP application need to store customer information as well to support fulfillment processes (scheduling, shipping, billing, etc), these data should be always synchronized from CRM service.

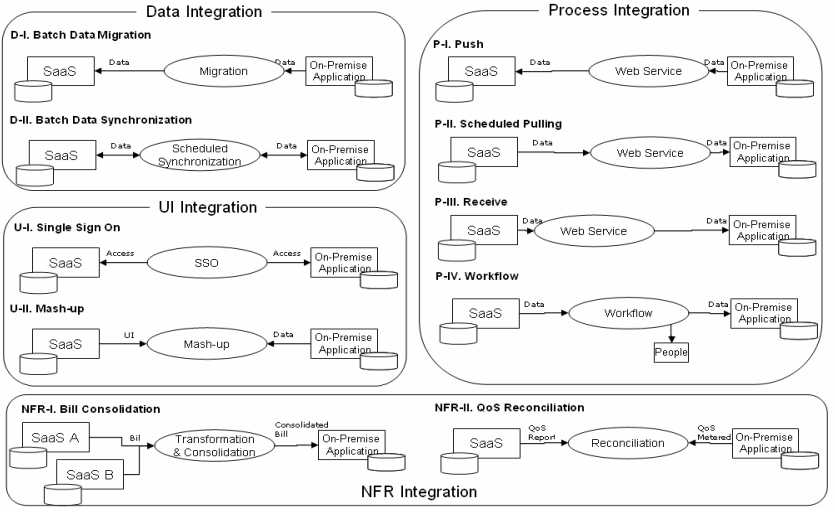

Fig. 2. SaaS Integration Common Patterns

2.2 SaaS Integration Non-Functional Requirements(NFR) and Patterns

SaaS services can be treated as Web Services from both macro level (services delivered over web) and micro level (leverage web services technologies to support integration). Most the NFR requirements brought by Web Services exist in SaaS domain as well, e.g. Security and Privacy. Here we point out the following three important requirements in the integration point of view.

a) Security and privacy

In most cases, all the SaaS subscribers' business data are centrally stored and managed by SaaS provider in a remote side over Internet. In the integration scenario, business data of every SaaS subscribers flow back and forth among the SaaS service and their on-premise applications over Internet. The integration technology should guarantee the subscriber's data should not be hacked and accessed by any third party.

b) Bill reporting and management

SaaS services are charged by usage. A bill is usually issued to the SaaS subscriber in certain timeframe by SaaS provider. As different SaaS providers issue different bills in terms of format and delivery method, the ideal integration scenario related with bill is illustrated in pattern NFR-I: different bills from different SaaS providers use same

format or can be transformed into same format, then could be centrally managed and fed into subscriber's finance and accounting application.

c) QoS reporting and reconciliation with SLA

SLA is usually included in a SaaS service contract between SaaS provider and SaaS subscriber. SLA often states the QoS related performance indicators, e.g. availability, response time. Most SaaS providers do provide QoS reports, however the reports are generated from service provider point of view only. As shown in pattern NFR-II, if the QoS of the SaaS services can be metered by the SaaS subscriber and generate report from consumer point view, then the QoS can be reconciled between service provider and service consumer so as to guarantee the SLA fulfillment.

2.3 SaaS Integration Design and Development Requirements

As illustrated in the following figure, the SaaS integration design and development process starts from business process review to analyze the key functional and NFR requirements; based on which to design and implement UI, process, data and NFR related integrations; then migrate/populate related data from master data source; finally test and go on production. This process is similar as the traditional application integration. However there are several SaaS unique issues we highlight as follows.

Fig. 3. SaaS Integration Design and Development Process

a) SaaS related policies' visibility

SaaS services have many policies which should be considered and utilized during the integration design and development process, e.g. configuration and customization policies. SaaS service usually serves many customers in multi-tenancy mode. Most customers usually have personalized requirements on the SaaS service. However the business model of SaaS is fundamentally about economic scale, which can only allow service configuration and customization within certain scope supported by self-service mode. Therefore the integration specialist should be instructed for the service configuration and customization policies during the integration design and development lifecycle. However, there is no related industry standard to support the definition of service's configuration and customization policies. Furthermore, to support those SaaS NFR integration requirements, the NFR policies of SaaS should be visible in the integration design and development environment.

b) Accommodation of different SaaS services in a unified environment

Currently different SaaS vendors have different toolkits to support integration, and System Integrators (SI) use different tools as well [4, 17]. However, a unified tooling environment can standardize the SaaS integration approach, so as to improve integration productivity, efficiency and accelerate the SaaS adoption accordingly. Since most SaaS services adopt Web Services technologies, it provides a very good foundation to accommodate different SaaS services in a unified tooling environment.

For most SaaS subscribers, functional integration requirements always have higher priority, NFR integration requirements can be value add features. In the following sections, a SaaS integration framework will be presented. This framework aims to streamline the SaaS integration design and development process for SI, supports the functional and NFR integration requirements accordingly.

3 SaaS Integration Framework

To address those SaaS integration requirements, in this section we introduce a SaaS integration framework reference architecture based on model driven integration approach [18], including SaaS-DL, tooling and runtime components.

3.1 SaaS-DL

SaaS can be treated as a kind of complex Web Services. Though Web Services Description Language (WSDL) can be used to describe interface related information, other information of SaaS services should be captured to support model driven SaaS integration. *WS-Policy [19] represents a set of specifications that describe the capabilities and constraints of the security (and other business) policies on intermediaries and end points, and how to associate policies with services and end points.* However, it does not address customization policy and some specific NFR policies clearly, which is strongly required in integration perspective. *In this section, w*e will introduce the design of SaaS-DL that is an extension to WSDL standards. The overall structure of SaaS-DL is depicted in figure 4. It leverages the WS-Policyattachment specification to bind itself to WSDL and XSD schemas*;* WSDL is also referenced in SaaS-DL, which describes the integration programming interfaces of the SaaS service. Three additional aspects are included. They are Customization Policy, Billing Policy, and Data Object Relationship Model.

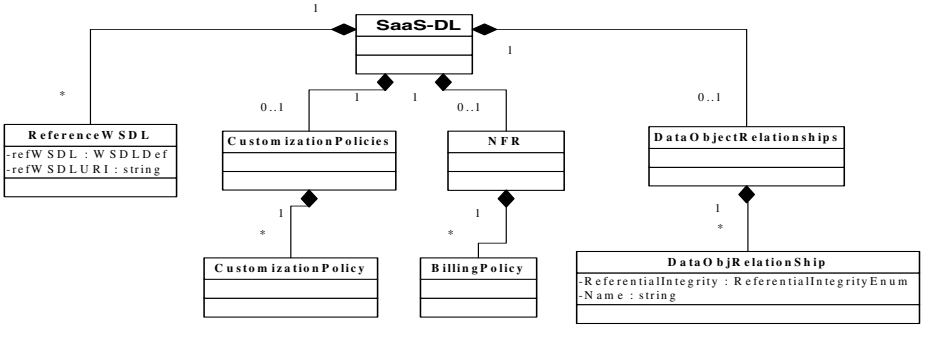

Fig. 4. Structure of SaaS-DL

Customization Policy
As analyzed in Section 2, SaaS services usually need to be customized to satisfy specific subscriber's requirements. Current Research topics on web services customization usually focus on semantic discovery or virtual wrappers [20, 21]; we propose a novel

approach by defining customization policy in SaaS-DL, and consuming it through SaaS integration lifecycle. Customization policy can be defined by SaaS provider, which annotates the SaaS service's customization capability to its subscribers; customization policy, customization process and related enablement technologies can streamline a standardized approach for the collaboration between providers and subscribers for the entire service customization lifecycle, the detailed design is discussed in paper [22].

Billing Policy
Billing is one of the most important NFR technologies required by SaaS. Many research works have been done on metering and accounting for Web Services [23, 24]. However, Web Services accounting is only one factor of SaaS billing concerns. Other factors should be considered, such as storage usage, and the membership types of SaaS subscribers. How to reasonably reflect the composite values, and consolidate different bills from different providers with different formats and styles, are important concerns. Therefore, a structuralized hierarchical model is proposed to organize the bill items and their relationships in a billing policy. Bill item is an atomic unit to describe the rule of billing, while the relationships in billing policy provide the power to specify how to compose a complex bill by combining atomic items recursively. The billing policy can be used to guide the metering of service usage so as to generate bill. Based on the policy, bill report structure can be standardized, so the bill reports from different SaaS providers can be easily consolidated into one bill for the SaaS subscriber.

Data Relationship Model
A SaaS service generally depicts a relatively complex service that involves many business objects (or data types) and their operations. As data relationship is not covered in WSDL, incorrect data manipulation can easily happen, for example, deleting one data object will bring major influence to another data. We propose to depict the data relationships of SaaS data in the SaaS-DL. This data relationship model in SaaS-DL is very much like that Entity Relationship(ER) diagram[25], where a data relationship is a triple of three elements: source object, target object, and its cardinality. Representing data relationships in SaaS-DL will also help SaaS customization process by analyzing and populating the impact of the customization to one data to another data.

3.2 Integration Framework

Here we introduce **SaaS integration accelerator** (SaaSia), which is reference architecture of SaaS integration framework. As shown in figure 5, the framework enables a collaborative integration environment for SaaS service provider and SI. It covers all the major aspects of the integration requirements and processes from design, development, deployment, and down to runtime support.

On the SaaS service provider side, the SaaS-DL Composer provides a tool for the service provider to describe the service information in a SaaS-DL, and then to publish it into the service registry to share with service subscribers. The Customization Engine provides a standardized interface to fulfill the customization requests. Through validating, analyzing, and decomposition, the Customization Engine weaves these requests into existing SaaS services, updates its implementation/configuration, and dynamically loads the upgraded service for the requestor. The NFR Reporting Service offers web services interfaces for subscribers to access NFR reports, e.g. bill, QoS report.

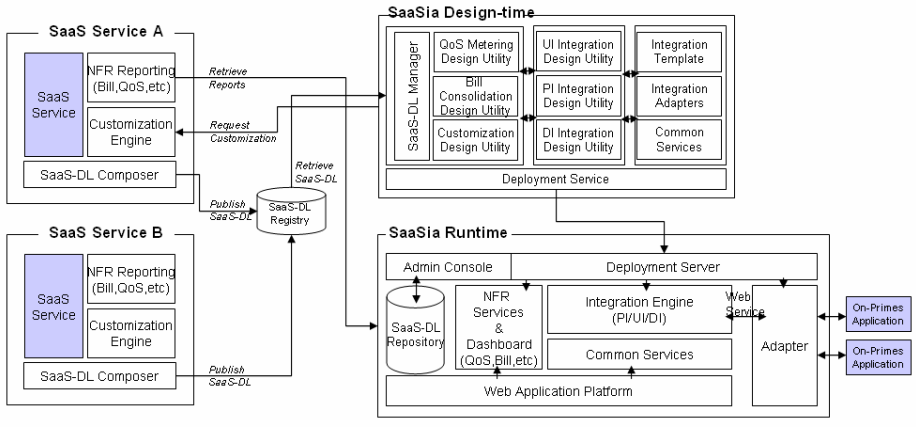

Fig. 5. SaaSia Framework Reference Architecture

On the SaaS service subscriber side, SI can use the design-time integration tool to design/develop the SaaS integration artifacts and deploy them into runtime environment, then automates the execution of integration logics on the runtime platform to meet its customer's needs. The SaaS-DL Manager component retrieves SaaS-DL from service registry and manages it in local repository. Customization Design Utility provides the customization controller for SI to handle the customization requirements in the whole integration lifecycle. The requirements are controlled within the scope defined by customization policy in SaaS-DL. The utility generates customization requests and send to SaaS service' Customization Engine to fulfill. The Bill Consolidation Design Utility can be used to design how the bills are retrieved from SaaS providers and then consolidated as one bill. QoS Metering Design Utility is used to define how the SaaS service's usage is metered so as to generate QoS report locally. Beside the core components introduced above, SaaSia design-time leverages common PI/UI/DI Design Utilities(e.g. BPEL[26]). The Deployment Service packages all the integration artifacts and deploys the package to runtime environment.

SaaSia runtime provides fundamental services and integration capabilities from different perspectives. The SaaS Repository manages the SaaS-DL and provides interface for runtime usage. NFR Services include two key services: QoS Metering service meters the SaaS services' utilization(transaction numbers, response time, exception rate, etc); the Bill Retrieval and Consolidation service fetches the bill reports from SaaS provider, transform and consolidate multiple bills into one integrated bill. The NFR Dashboard component provides a visualized presentation about the bill and service utilization information. The Adaptor is a runtime framework to enable the integration with on-premise application using required network protocol and programming interfaces.

The SaaSia runtime architecture can be implemented in two different deployment modes illustrated in figure 6. If the SaaS subscribers have strong integration requirements about security and privacy, they should select the local deployment mode which provides dedicated SaaSia runtime; If the SaaS subscribers prefer to get the integration capability as hosted services, they should use the remote deployment mode. In this

mode, SaaSia adapter should be deployed at SaaS subscriber's premise to connect with on-premise application, the functional and NFR integration logics should be deployed to a hosted SaaS integration hub which provides integration services in multi-tenancy mode for many SaaS subscribers.

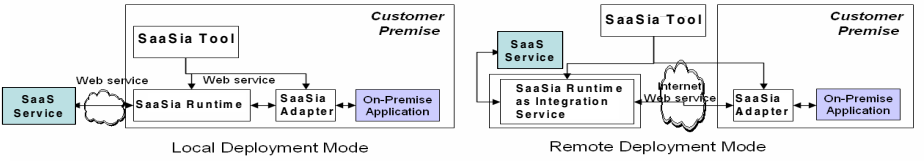

Fig. 6. SaaSia Deployment Mode

3.3 SaaSia Prototype

According to the reference architecture, a SaaSia prototype is built. The tooling prototype embraces the lightweight and open Eclipse platform. It also benefits from the full functionalities brought by the Eclipse projects, e.g., web tool by Eclipse WTP, data transformation by Eclipse DTP, dashboard by Eclipse BIRT, and BPEL programming by Eclipse BPEL [27]. There are also pre-built assets to accelerate the integration design/development, including Common Services (e.g., scheduling, logging), Integration Adapters(e.g. Adapter for SAP, Quickbooks) and Integration Templates(e.g. CRM opportunity to ERP order fulfillment). As illustrated in figure 7, SaaS-DL Manager, SaaS Customization, NFR Dashboard Design Utilities and Deployment Utility can integrate with these Eclipse components as a SaaS integration design and development toolkit.

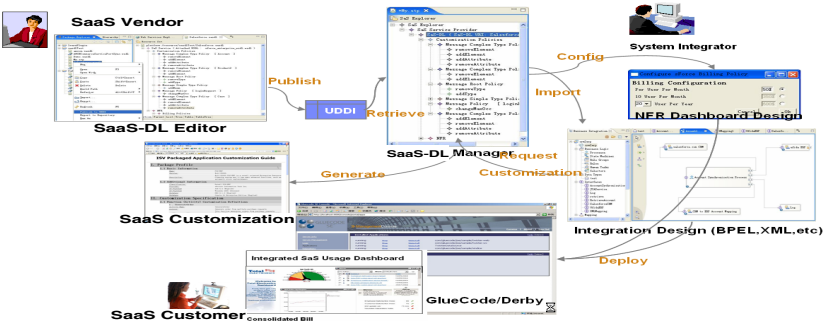

Fig. 7. SaaSia Design-Time Prototype

The SaaSia Runtime prototype adopts the local deployment mode. It focuses on lightweight integration capability at SaaS subscriber premise environment. SaaS runtime is an integrated platform built by leveraging open source and existed components as much as possible. As shown in figure 8, SaaS runtime provides three key modules: administration console, integration platform, and SaaS utilization dashboard. The

integration module provides integration related capabilities such as BPEL engine, ETL engine, and legacy application integration through JCA adaptor. SaaS NFR dashboard demonstrates the result of SaaS usage metering and bill consolidation. Administration components offer the SSO and SaaS-DL management services.

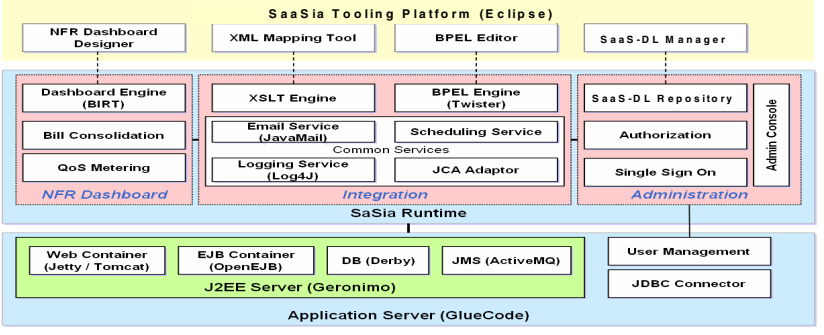

Fig. 8. SaaSia Prototype Architecture

4 Case Study

In this section, a case about integrating CRM SaaS service and ERP on-premise application is studied. The customer company has hundreds of employees and 4 offices in different cities in China. An ERP application has been deployed for several years to support manufacturing related business. Recently the company subscribed a SaaS CRM service to better support their customer related business. Though they started to use the

Table 1. Business Requirements and corresponding pattern and actions

Business Requirements	Pattern	Integration Actions
"Product" information synchronization from ERP to CRM service; "Account" information synchronization from CRM service to ERP	D-II	Customize the "Product" & "Account" data structure on CRM service to map with ERP; Leverage Scheduling service and CRM service & ERP application api to synchronize data
Pass new "Order" information from CRM service to ERP	P-III	Customize the "Product" & "Account" data structure on CRM service to map with ERP; Leverage Scheduling service and CRM service & ERP application api to synchronize data
Pass new "Shipping Notice" and "Invoice" information from ERP to CRM service and update original "Order"'s status	P-I	Create new data structure "Shipping Notice" and "Invoice" and build relationship with "Order" using Order_ID; Develop new web service to feed the data into CRM services
Sales Person creates a new "Product" request according to customer's special requirements, the request will be sent to Product Manager to approve, and then feed into ERP system to guide fulfillment.	P-IV	Create a workflow and link the workflow with CRM service/ERP application api
Have an integrated user interface to access both CRM service & ERP application	U-I	Create a new web page to accommodate the ERP & CRM service with UI supported by SSO.
Collect the usage statistical information of the CRM service	NFR-II	Configure NFR dashboard based on web service metering capability

SaaS service as a standalone application, they eventually found that it had to be integrated with their on-premise ERP application. The detailed requirements, patterns applied and developed integration actions are listed in the following table1.

As illustrated in figure 9, the requirements listed above have been fulfilled by SaaSia prototype technologies. There are two important lessons gained through our practice:

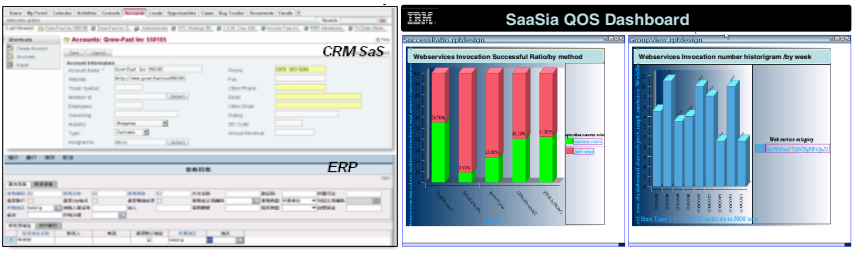

Fig. 9. Integrated Solution based on CRM SaaS and ERP on-Premise Application

a) Most SaaS services don't provide programmatic interfaces for customer to retrieve QoS and Billing reports. Different SaaS services use their own tools to describe customization capability and perform customization actions. So SaaS related standards should include these perspectives to benefit the SaaS growth.

b) As currently most SaaS services' subscribers are SMBs [6]. They strongly -expect integration to be done with very small footprint in agile way. The current SaaSia prototype is standard based, e.g. Eclipse, BPEL. But to gain SMB adoption we need to explore more lightweight approach including browser based integration tool and programming model based on Web 2.0 technologies [12, 28].

5 Conclusions and Future Work

In this paper, we analyzed the key requirements for SaaS integration and presented several integration patterns. A SaaS integration framework, SaaSia, is proposed to address those requirements. Also a prototype and corresponding case study is introduced. We learned two valuable lessons. Firstly, most SaaS integration functional requirements can be fulfilled by existing SOA integration technologies [5]; Secondly SaaS involves some NFR requirements which should be addressed by extending exiting integration technologies. We plan to pursue future work in two directions. As there lacks of industry standards to streamline SaaS integration, we will conduct more research around the concept of SaaS-DL [22] in Enterprise Application Integration, leverage and Enhance BPEL or ESB; we will also dive into the latest Web2.0 technology [12, 28], e.g. apply SaaS-DL in Mash-up description languages, to explore a more lightweight and generic SaaS integration platform for SMB.

References

[1] Knorr, E.: Software as a Service: The Next Big Thing, http://www.infoworld.com/article/06/03/20/76103_12FEsaas_1.html

[2] Summit Strategy Report: The Future of Software as Service-And the Partners ISVs will Need to Get There (2004)
[3] Web Site, http://www.employease.com
[4] Web Site: Salesforce.com AppExchange, [Online]: http://www.salesforce.com
[5] Newcomer, E., Lomow, G.: Understanding SOA with Web Services. Addison-Wesley, Reading (2004)
[6] Baumol, W.: Small Firms: Why Market-Driven Innovation Can't Get Along Without Them., The Small Business Economy: A Report to the President, Ch. 8, pp. 183–206 (2005)
[7] Web Site: Mashups and the Web as Platform, http://www.programmableweb.com/
[8] AMR Research Report: Software as a Service: Managing Buyer Expectations as We Pass the Tipping Point from Novelty to Necessity (2005)
[9] IDC report: Software as a Service in the Mid-market: Adoption Trends and Customer Preferences (2004)
[10] Web Site, [Online]: http://www.jamcracker.com
[11] Web Site, SaaS Showcase, [Online]: http://www-19.lotus.com/wps/portal/showcase/SaaS
[12] Seltsikas, P., Currie, W.L.: Evaluating The Application Service Provider (ASP) Business Model: The Challenge of Integration. In: Proceedings of the 35th Hawaii International Conference on System Sciences (2002)
[13] Elfatatry, A.: Software As A Service: A Negotiation Perspective. In: COMPSAC'02. Proceedings of the 26th Annual International Computer Software and Applications Conference (2002)
[14] Turne, M.: turning Software into a Service, Computer (October 2003)
[15] O'Reilly: What is Web 2.0, Design Patterns and Business Models for the Next Generation of Software (2005)
[16] Web Site: NetSuite Small Business, [Online]: http://www.netsuite.com/
[17] Web Site, [Online] available: http://www.aboveall.com
[18] OMG: An Architecture for Modeling, http://www.omg.org/mda
[19] W3C WS-Policy standard: http://schemas.xmlsoap.org/ws/2004/09/policy/
[20] Mandell, D., McIlrait, S.: Automating Web Service Discovery, Customization, and Semantic Translation with a Semantic Discovery Service. The Twelfth International World Wide Web (2003) (reference 26)
[21] Rykowski, J.: Virtual Web Services - Application of Software Agents to Personalization of Web Services. In: 6th International Conference on Electronic Commerce ICEC 2004: Engineering the New Landscape, pp. 419–428. ACM Publishers, New York (2004)
[22] Zhang, K., Sun, W., Zhang, X., Liang, Hq., Huang, Y., Liu, X.: A Policy-Driven Approach for SaaS Customization. In: The 9th IEEE Conference on E-Commerce Technology, IEEE Computer Society Press, Los Alamitos (2007)
[23] Aboda, B., Arkko, J., Harrington, D.: Introduction to Accounting Management, RFC2975 (October 2000)
[24] Agarwal, V., Karnik, N., Kumar, A.: Metering and Accounting for Composite e-Services. In: CEC'03. Proceedings of the IEEE International Conference on E-Commerce, IEEE Computer Society Press, Los Alamitos (2003)
[25] Web Site, [Online] available, http://www.umsl.edu/~sauter/analysis/er/er_intro.html
[26] IBM: BEA Systems, Microsoft, SAP AG, Siebel Systems, Business Process Execution Language for Web Services version 1.1
[27] Web Site, [Online] available, http://www.eclipse.org
[28] Gross, C.: Ajax Patterns and Best Practices, Apress (2006)
[29] Web Site, OpenKapow, http://openkapow.com/
[30] Ottinger, J.: Software as a Service Integration via Mule, http://www.theserverside.com/news/thread.tss?thread_id=44456