J Supercomput
DOI 10.1007/s11227-016-1625-y

SaaS-based enterprise application integration approach and case study

BeiLie Wang[1] · **Hui Liu**[2] · **Jie Song**[1]

Abstract Software-as-a-Service (SaaS) has been well studied, and it is being adopted at a very fast pace. Enterprise application integration (EAI) is the key factor in many enterprises. Traditionally, SOA technique can be applied to EAI, especially the application of Web Service. In this paper, we propose that SaaS can also be applied to the field of EAI. Based on this, we propose the SaaS-based EAI approach which rebuilds one analog of legacy applications into SaaS architecture makes the rest applications be configured, and further solves the EAI problem. Besides, we also update the traditional SaaS maturity model, discuss relationships between SOA and SaaS in EAI, and explain how to further integrate SaaS applications by SOA. We implement an Eclipse plug-in which can introduce SaaS capabilities into traditional Web application automatically. Finally, the proposed approach is proved to be effective through a case study.

Keywords Cloud computing · SaaS · SOA · Enterprise application integration (EAI)

1 Introduction

Nowadays, Software-as-a-Service (SaaS) has become one of the most popular computing paradigms in the field of cloud computing. With the rapid development of the Internet technology and the maturity of Web-based applications, SaaS, which is known as a completely innovative model of software, is being adopted to develop more

✉ BeiLie Wang
133268682@qq.com; wangbl@swc.neu.edu.cn

1 Software College, Northeastern University, Shenyang, China

2 School of Materials and Metallurgy, Northeastern University, Shenyang, China

Published online: 27 January 2016

and more Web applications. People believe that the cloud computing is essentially the large-scale delivery of services to end users over Internet, and SaaS is a software delivery model that allows customers to shift and restrict their IT responsibilities, rather than a application integration model.

The computer infrastructure of a typical today's enterprise can be seen as a software ecosystem that involves several complementary applications purchased from different providers or built at home [1]. A recurrent challenge is to make these applications interoperate with each other to keep their data synchronized or to create a new piece of functionality, this problem is known as enterprise application integration (EAI). EAI is the process of bringing two application programs together with data or a function from them [2]. When these programs already exist, the process is sometimes implemented using middleware, either packaged by a vendor or written by a custom. We define the "analogs" is the term for the applications are similar in function. Some EAI tasks integrate these analogs together, and treat them as a uniform one. There is a common challenge that is the communication between analogs because some of them may be developed in different languages or based on heterogeneous data. In this case, EAI is urgently on demand. In general, for applications, the use of object-oriented programming, actual or de facto standard development tools and interfaces (such as Java or .NET) will be beneficial for another new application to be easily integrated. The Extensible Markup Language (XML) promises to serve as a tool for exchanging data amongst disparate programs in a standard way. Traditionally, there are two patterns to implement EAI:

- Mediation: the Mediated system acts as broker between multiple applications.
- Federation: the Federated system acts as the overarching facade across multiple applications.

Well-known Service-Oriented Architecture (SOA) can be adopted in EAI, especially the application of Web Service [3]. As organizations strive to build loosely coupled systems based on SOA principles, the issues about data redundancy, quality and consistency are exacerbated and become significant barriers to successful integration. Most of EAI-related previous works are based on mediation or federation, in which SOA architecture is adopted as the implementation of interoperability [4–6]. But there exists some shortcomings in the two patterns. First, wide knowledge is required when integrating the heterogeneous applications; second, the wrappers for interoperability will cause high complexity in both interface and logic; third, the cost of maintaining multiple applications is still high even if they are integrated. For example, most EAI projects usually start off as point-to-point efforts. It will create lots of connections as the number of applications increases so that the exposed interfaces are unmanageable.

In this paper, we propose an approach that can partly solve the EAI problem by SaaS Paradigm. Sometimes, the EAI problems can be solved by customizing a single instance for multiple tenants, instead of multiple applications. We SaaSify (terms for converting an application into SaaS paradigm) one of analogues in legacy applications, making it can be customized to be the other applications by clients, and then we update the SaaS maturity model for such an approach; we explain that SOA could also be used for integrating SaaSified applications. We propose a SaaSify tool for introducing

SaaS capabilities to a certain Web application automatically. Finally, the proposed approach is proved to be effective by a case study.

2 Materials and methods

SaaS has been widely studied in lots of fields. Karabulut and Nassi [7] provide a secure and trusted service consumption environment by creating a fine-grained operating cost model for SaaS. Liu et al. [8] analyse methods and communication infrastructure, which enables distributed SaaS applications over the data network. Yan and Zhang [9] enable progressive migration of multi-version applications in SaaS through schemas. Lu and Sun [10] create a fitness evaluation model to estimate whether SaaS is suitable for the information system evaluated. The above-mentioned works of SaaS do not apply to EAI.

System integration can be performed on different levels [11]. Following the work described in [12]: integration on the data source level provides a unified view on heterogeneous data sources [13]; integration on the business logic level unifies different implementations of business logic, each using its own data sources, under a common user interface. The most prominent approaches on this level are semantic web services [14] and ontology-based agents [15]; integration on the user interface level unifies different user interfaces in one common system, e.g. a portal or a plugin-based user interface. The proposed SaaSified approach can integrate the application in both three levels but performing in the interface level is much easier. Most researches in EAI are focused on loose coupling, easy expansion and flexibility issues. He et al. [4] propose an SOA-based application integration solution through JavaEE approach. Ji [5] proposes an application integration framework on the basis of Web Services. Chen et al. [6,16,17] also adopt the SOA and Web Services. Comparing these methods with SaaSified approach proposed in this paper, there are some shortcomings such as requiring wide knowledge, high complexity of interfaces and high cost.

3 SaaS approach

And SaaS maturity model will be updated to meet practical requirements of EAI. We will also discuss relationships between SaaS and SOA, and then explain how to integrate SaaS applications by SOA.

3.1 SaaS-based EAI approach

Presently, SOA-based EAI has been widely used. To achieve interoperability, SOA-based approach encapsulates original applications into services on the basis of same standards, which is the way for services to interoperate (see Fig. 1).

Figure 2 shows how to achieve EAI through SaaS solution. First, an appropriate analog is selected as the target analog, and other analogs can be configured in accordance with the target analog. As shown in Fig. 2, applications A and B are similar. In this case, A is selected as the target analog; B can be configured in accordance with A.

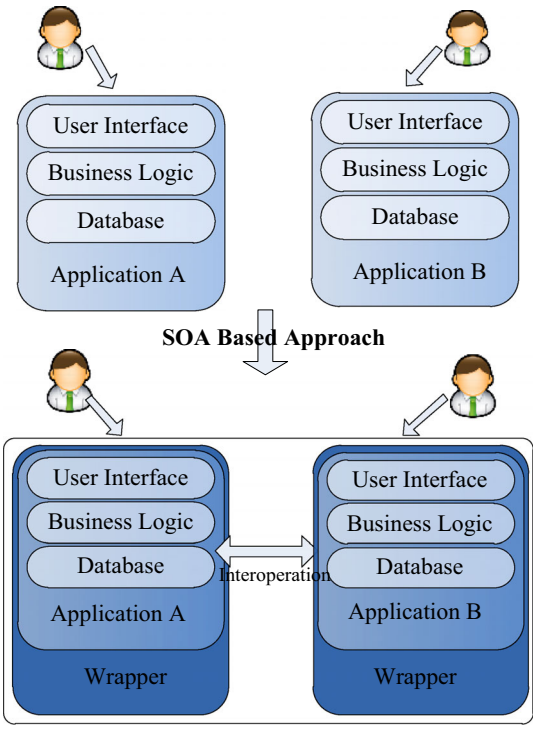


Fig. 1 SOA-based EAI approach

Besides, it makes no difference for tenants to use application B in the form of SaaS, compared directly with the use of original application B.

Some problems in EAI, including data format problem, can be solved through the SaaS-based approach we proposed. SaaS-based approach focuses on configurability whilst SOA-based approach focuses on interoperability. The later has some shortcomings, such as it requires programmers to have wide knowledge, and its interface has high complexity and cost. The business concept of SaaS has to change if the SaaS-based EAI approach is adopted in an enterprise, SaaS application will be deployed in the enterprise's context instead of the vendor's. In this situation, there is no need for the enterprise to worry about data security, SLA and sustainability of SaaS. At the same time, SaaS applications will cut down the cost of maintaining multiple applications in each branch.

Configurability of SaaS is the key point here. Thus, we need to update maturity model of SaaS to evaluate its configurability. Details about the updated maturity model are shown in the following part.

3.2 Updating SaaS maturity model

In this section, we update the SaaS maturity model for clear and detailed evaluating EAI capabilities of SaaS applications. Mladen [18,19] considers four levels of SaaS maturity models related to support of tenants vis-a-vis instances of software solution

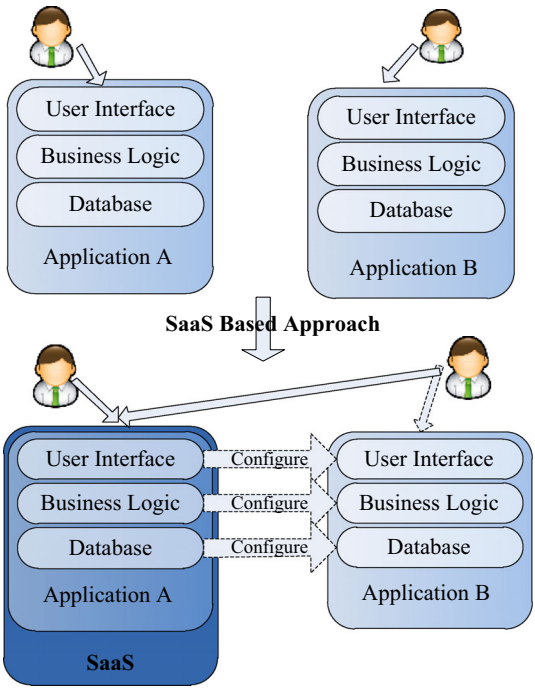


Fig. 2 SaaS-based EAI approach

and database. For the page limitation, the details of these four levels are abbreviative. Due to the SaaS solution for EAI, the configurability of the SaaSified application should be paid close attention to. So we add another dimension to SaaS maturity model and name it as "Configuration level". Thus, the maturity model is changed as a two-dimension model: the one dimension is the original Level 2, Level 3 and Level 4; the new dimension "Configuration level" divides them into three levels individually. Software systems most often consist of three layers: the data source layer, the business logic layer, and the user interface layer [20], so that the Configuration levels are Interface Level, Interface-store Level, Interface-store-logic Level.

User interface configurability makes the system interface to be easily configured according to tenants' expectation. It consists of two aspects including configurability of interface and page content. Configurability of interface means tenants can adjust the appearance according to their preferences, such as position, number and size of interface elements. Configurability of page content means tenants can adjust service content according to their preferences. SaaS applications of multi-tenants are used by different tenants under the same instance. Thus, the system should meet tenants' different requirements on data issues. There are mainly three solutions in data configurability, including customized fields, pre-assigned fields and name–value pairs. Business logic configurability means tenants can adjust and modify the process and approach of service's implementation according to their requirements.

There are two reasons why the updated SaaS maturity model is defined in the order of interface, data and logic. First, configurability of user interface is the basic requirement

in SaaS. As the entrance of SaaS applications, it is easy to configure the user interface, compared with that of data and logic. Second, changes of data configurability will affect logic layer. The business logic of a SaaS application involves operation on data. Thus, it is clear that logic configurability is more complex than data configurability. With the updated SaaS maturity model, SaaS-based EAI approach has more feasibility, and SaaS application can be configured from the original ones.

3.3 Integrating SaaS applications by SOA

In this section, we discuss how to integrate the SaaS application. As mentioned before, applications could be integrated by SOA and SaaS, but there is a new challenge, that is, the SaaS applications from different vendors could also be integrated to implement the coarser-grained business process. Traditional applications? integration is the integration of a system. On the contrary, SaaS is a package of services. The integration of SaaS applications is a higher level of application integration. This kind of integration can be solved through the SOA technique. To achieve the integration of SaaS services through SOA approach, large numbers of small SaaS services are integrated together so as to provide more abstract and coarse-grained software services.

The difference between SaaS and SOA is that, in sum, the former is a software-delivery model whereas the latter is a software-construction model [21]. SaaS, also known as the "subscription software", essentially separates software ownership from the user—the owner is a vendor who hosts the software and lets the user execute it on demand through some form of client-side architecture via the Internet or an intranet. In an SOA model, the constituent components of the software system are reusable services. A collection of services interacts with each other through standard interfaces and communication protocols. SOA promises to fundamentally change the way we build internal systems as well as the way internal and external systems interact. SOA and SaaS both define the interaction model amongst its components, and they both need techniques (e.g. Web Services) to achieve their interaction model. However, SOA consists of service components whilst the components of SaaS are not all services.

Although there are significant differences, SaaS and SOA are closely related and complementary for large-scale information systems. Software can be treated as a service and published through SaaS, and then SaaS provides components that SOA uses. Moreover, SOA can search and use the published services, and these service components can be combined to rapidly create new software system. In this way, the cost of software design and development can get significantly reduced. The new system can also be treated as a service, and then published through SaaS. In addition, SaaS allows a software system to deliver with multiple different granularity levels of services, and therefore provides more services of different complexity to build SOA-based system. In a word, SaaS provides components that are able to be used by SOA, and SOA can help to implement SaaS more quickly.

As shown in Fig. 3, existing software systems (i.e. system 1 to n) could be accordingly converted to standard Web Services, and registered to Service Registry to provide service description that is available for search. Service request could be sent out by Service Consumer (e.g. combined request shown in Fig. 3) and construct new soft-

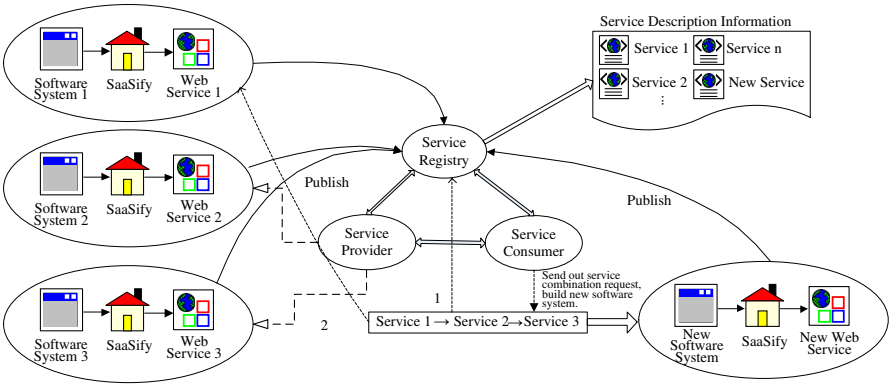


Fig. 3 Relationships between SOA and SaaS

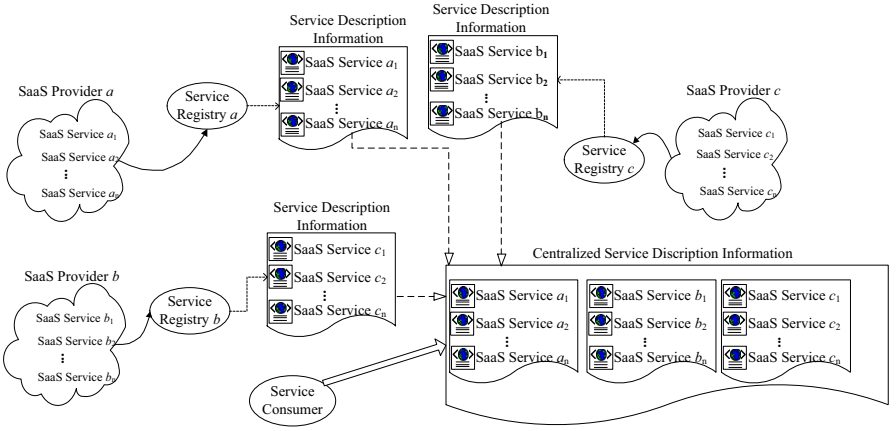


Fig. 4 Registration information federated cloud for SaaS services

ware system (such as service 1 →service 2 →service 3). For simplifying, Fig. 3 only represents the searching process of service 1 and the binding process, similar to other services. The new constructed software system could also be SaaSified and published to Service Registry so that it is able to be used by Service Consumer. Thus, existing service components can be combined to rapidly create new software systems and services.

So far Service Registries of different SaaS service providers are dispersed, and maintain their separate service registration information, respectively [22]. Therefore, by the way of collecting service registration information of different Service Registries together, we can achieve the purpose of unified management and then build service registration information federation cloud. In this way, the efficiency of service discovery can be greatly improved, which is helpful for Service Consumers to find service combinations that meet their needs transparently. Meanwhile, service discovery-based applications such as combination and selection of services can be further improved, which is shown in Fig. 4.

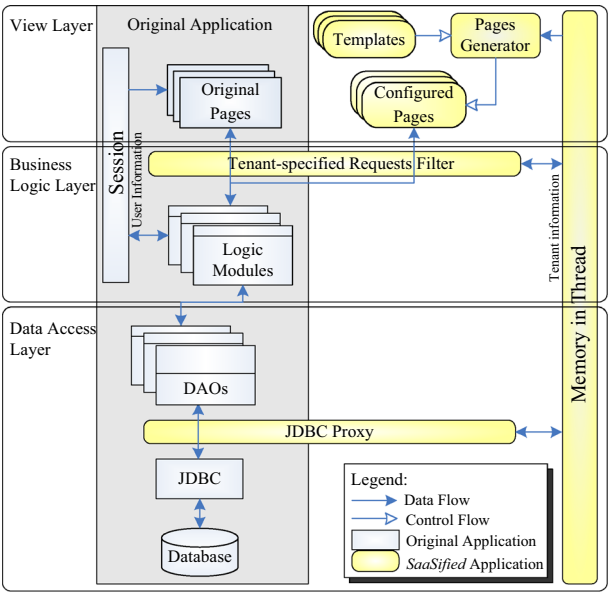


Fig. 5 The extended architecture of the target application

4 SaaSify tool

In this paper, for easily introducing the configuration capabilities to no-SaaS applications, we proposed an automatic SaaSify tool which adopts several technologies and extends the architecture of Web applications. Figure 5 shows the modified architecture of the SaaSified application, which extends the original one without modifying the source code. The extensions are transparent to the original application, thus the original architecture is reinforced but not refactored.

The expanded SaaSify architecture is still divided into three layers [20], including View Layer, Business Logic Layer and Data Access Layer (see Fig. 5). First, user logins through user interfaces in the view layer. Then, user's information is delivered to business logic layer, in the meantime tenant information is intercepted by the filter and stored in memory thread in the process of login. Then, page builder generates corresponding configuration page on the basis of tenant information stored in memory thread. After that, user information is obtained from the session in business logic layer. Users can use the functions of their respective tenants which have been ordered. Whilst connecting to the database (i.e. the process of executing SQL commands), JDBC proxy is responsible for adding tenant restrictions to SQL commands.

There are two kinds of configurabilities in SaaS paradigm, the first one is "configurable interfaces", and the other one is "configurable functions". We do not use a priori knowledge to provide functions of the original knowledge to SaaSify tool, so the tool extracts the function list by analysing buttons and links in pages. Above all, web pages should display according to tenants configurations. First, the position and colour of page elements should be in accord with tenants' configurations; second,

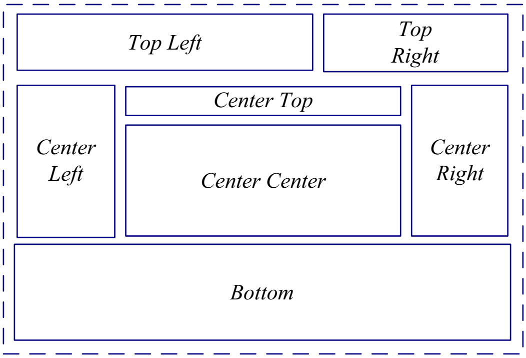


Fig. 6 Example of dividing page blocks

tenants can only see the links and buttons whose corresponding functions have been paid. An operation named configurable interfaces is adopted to support the former, and another operation named configurable functions' is adopted to support the later.

To implement "configurable interfaces", we divide the original page into blocks, then it is possible to select and regroup this blocks by configurations. The "configurable interfaces" follow seven steps:

Step 1: Scan the HTML sources of pages
Step 2: Divide the page into visual blocks according to the visual effect as shown in Fig. 6, which is concluded by us and satisfied most of web pages layouts.
Step 3: Analyse the HTML code of each visual block, extract them into page blocks;
Step 4: Number these blocks, re-implement page blocks, and then distinguish them from each other by adding some HTML invisible tags such as < div >.
Step 5: Assign the page blocks to tenants' configuration.
Step 6: Implement algorithm for freely compositing page blocks into page by script language.
Step 7: When a user visits a page, the page generator retrieves the tenant-info which the user belongs to, and then dynamically regroups the pages according to tenant's configuration.

To implement "configurable functions", we extract URL from links and buttons in each page block. Then, it is possible to show or hide these elements selected by configurations. The operations follow seven steps:

Step 1: Scan all links and buttons of each page blocks, extracting their URLs.
Step 2: Analyse these URLs, retrieve a proper name from each URL as the function name. In practice, these names are not intuitive. Therefore, we provide an interface to let the administrator replace these namespaces with intuitive ones. Some non-functional URLs are also be excluded by administrator.
Step 3: Make these links and buttons independent from pages by adding some HTML invisible tags such as <div>.
Step 4: Assign the functions to tenants' configuration.
Step 5: Implement algorithm for freely show or hide these links and buttons by script language.

Step 6: When a user visits a page, the page generator retrieves the tenant-info which the user belongs to, and dynamically shows the links and buttons whose corresponding functions have been paid by current tenant.
Step 7: Introduce a request filter to the web server, for example, add a filter servlet to "web.xml" in JavaEE application, and make sure that the URLs are requested, even they are unpaid. Because they will be discarded.

Many other technologies are also used for implementing SaaSifying. For example, metadata are used for implementing data isolation. SQL parser is used for scanning SQL script. Dom4j is used when page templates are generated. Log4j is used for log whilst SaaSifying. These technologies are already widely accepted and will not be introduced in this paper. We have developed an Eclipse plug-in of SaaSify tool. Currently, it can only SaaSify the JavaEE-based web application. The meta-information can be provided by the configurations: the SQL_Config_File describes the database information of web application; the Function_Points_File lists all functions of web application; the JDBC_Property specifies the location of JDBC driver which could be updated; the Login_Page specifies the login page of web application, it will be modified by introducing the tenant information; the JSP_Directory specifies the root directory of JSP pages; the Login_Path specifies the login URL of web application to be transformed. This plug-in is success to SaaSify the Java Pet Store which is a well-known standard application given by Sun Co., and designed to illustrate how the Java Enterprise Edition 5 Platform can be used.

5 Case study

In this section, a case study will be used to show that the proposed approach is effective.

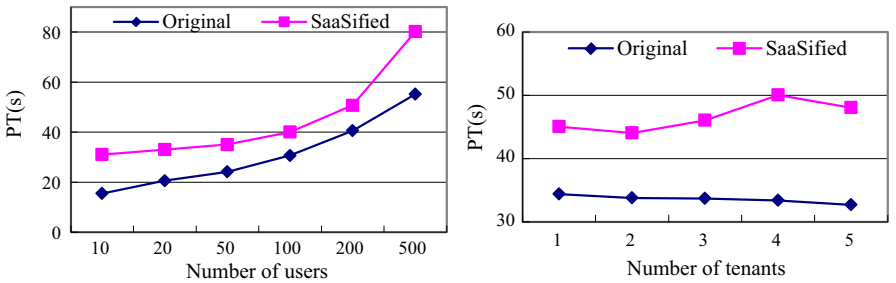
5.1 Background

In DanDong Power Equipment Group, there are three vote systems. One is subsystem of ERP system, another one is individual vote system as the legacy, and the rest one is a local system used by one of branches in another city. The functions of three systems are similar but have different focuses and different basis. The Group wants to integrate the three systems for higher functionality and getting statistical data more easily. In this case, our proposed SaaSified approach is suitable for the Group.

According to the rules mentioned in Sect. 2, the subsystem of ERP is chosen as the one to be SaaSified. In such cases, two systems are integrated for quantitative analysis of the performance of proposed SaaSified approach. After that, a comparison with SOA solution is discussed.

5.2 System performance

To get the atomic performance comparison, we design a single user condition and evaluate the process times (PTs) of SaaSified system and original system under a voting process (several requests and responses).



(a) Fixed Tenants **(b)** Fixed Users

Fig. 7 Experiment results of process time

The average PTs we got from the experiment are "7.8 s for the Original and 9.2 s for the SaaSified (time of user operations are not included)". It is obvious that the SaaSified takes a little more time than the Original. By analysing, the additional time is consumed by the following process:

- When the user logs on the system, the tenant-info is selected from database and loaded to current thread.
- When searching a record, the tenant-info is retrieved from current thread and queried as a condition.

Though the PT for SaaSified application is a little higher than the Original one, it is neglectable comparing to the functional improvement. In the following experiments, we use LoadRunner [10] to simulate the concurrent scenario for evaluating the three measures. In every experiment, we design the following two methods.

- Fixed Tenants: Under the 5 tenants, compare the performance of two applications in N concurrent users (N/5 users per tenant), N is 10, 20, 50, 100, 200 and 500.
- Fixed Users: Under the 100 concurrent users, compare the performance of two applications in M tenants (100/M users per tenant), M is 1, 2, 3, 4 and 5.

In this experiment, we study the average PT of server when running two applications in Fixed Tenants and Fixed Users, and compare them. The results are shown in Fig. 7.

From Fig. 7, the conclusions can be drawn that:

- In both two methods, PT of the SaaSified is larger than that of the Original, it matches the results of the first experiment. The additional time is consumed to process tenant-info.
- As is shown in Fig. 7a, PTs of both the SaaSified and the Original keep stable at first, then increase apparently when the users are more than 100. By monitoring the Tomcat server, we found that some exceptions of database connection occurred; the database reached the bottleneck. This is the main reason for the increase of PT.
- From Fig. 7b, when the tenants increase, PT of the SaaSified keeps stable whilst PT of the Original decreases. It is because that the deployed architecture of SaaSified is not changed with tenants, so as its PT. On the contrary to the Original, the concurrent users per tenant (per instance in Tomcat server) decrease when the tenant increases, so the process capacity of each instance improves. Thus, the PT of Original tends to reduce.

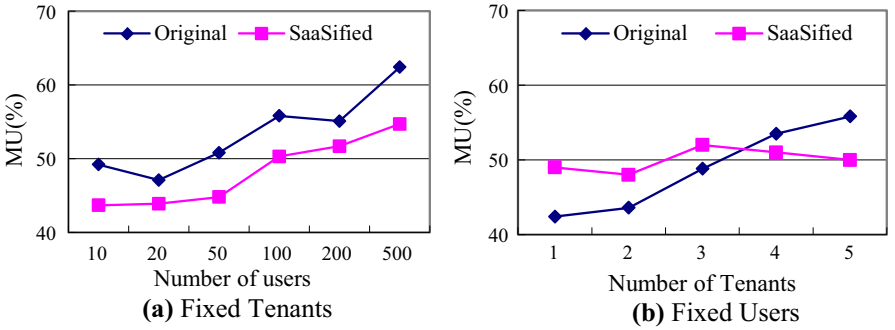


Fig. 8 Experiment results of memory usage

In this experiment, we study the memory usage (MU) of server whilst running two applications in Fixed Tenants and Fixed Users, and compare them. The results are shown in Fig. 8.

From Fig. 8, some conclusions can be drawn that:

- The overall trend of MU is similar to PT, in Fig. 8a, MUs of both the Original and the SaaSified increase when users increase. In Fig. 8b, when tenants increase, MU of the SaaSified keeps stable whilst that of Original increases. We do not explain it in detail.
- The advantage of SaaSified emerges when the number of tenants is more than three, because at that point, the memory to maintain instance of the Original is more than the memory to maintain tenant-info in SaaSified.

Summarizing all the experiments, several general conclusions are drawn and listed here:

- PT of the SaaSified is little higher than that of the Original. The additional cost on maintaining the tenant-info is worth the cost of SaaS functionality.
- PT and the MU of the SaaSified are apparently lower than those of the Original, because of the advantage of SaaSified architecture.

In all the above experiments, we prove that the SaaS-based Enterprise Application Integration approach is efficient and does not bring too much performance cost when SaaS capabilities are introduced. The experimental results are from an typical JavaEE-based ERP system in industry, thus the similar results could be also drawn.

5.3 Comparison

After vote system in ERP is SaaSified, it integrates the features of two original vote systems in user interface, data and logic. The two systems are integrated into the one in ERP system, thus all the users can configure the SaaS system to implement their original requirements and vote through a uniform interface; the vote data are no longer distributed; the statistical results can be calculated more easily; users from different branches are also classified well; EAI is well implemented. Table 1 compares the characters of the original vote systems and new SaaSified system.

Table 1 Quantitative analysis of SaaSified approach

System	Original system	SaaSified system	Legacy system
Feature	Count		One of the legacy systems is developed in .NET, and SqlServer is chosen to be the database. The total lines of codes in this system are nearly 5000 lines
JSP	10 files	21 files	
Java	19 files	18 files	
Configuration	5 files	5 files	
CSS	4 files	4 files	
JS	7 files	7 files	
Table	3 files	8 files	
Code	5300 lines	7200 lines	
Database: Oracle			

Table 2 Comparison of SOA and SaaS approaches

Comparison item	SOA approach	SaaS approach
Applied area	More	Less
Integration times	More than one	Once
Scalability	Low	High
Cost	Higher	Relatively lower
Required knowledge	More	Less

From both the case study, the data in Table 1, and SaaSified tool mentioned in Sect. 4, we know that after dealing with EAI by proposed SaaSified approach, (1) heterogeneous systems, for example Oracle and MS SqlServer, are well integrated; (2) systems do not expand too much; (3) most workload of re-architecting systems for SaaS are done by proposed tool automatically; (4) the proposed approach can cut down the cost of infrastructure, operation and maintenance if the SaaS system can reach the maturity Level-3. These prove that the proposed approach is effective and efficient.

Then, the differences between SOA and SaaS approaches for EAI are discussed in Table 2.

From Table 2 the conclusions can be drawn:

- Applied area: The SOA approach can be used for all kinds of legacy integration by service wrapping. The SaaS approach for EAI aims at analogues of legacy software in different branches of the same enterprise.
- Integration times: The SOA approach should be applied to each of the legacies that should be integrated. The SaaS approach could be applied only once, for the selected legacy, the rest legacies can be configured by their customs.
- Scalability: The SOA approach encapsulates the original applications as services according to the same standard, so that applications can interoperate with each

other, but they are not changed for supporting large numbers of users. The SaaS approach with Level-4 maturity can handle with the great scalability.
- Cost: The cost of SOA approach depends on the number of legacies that should be integrated and potential investment on infrastructure and maintenance that may be high. The cost of SaaS approach depends on the difficulty of re-architecting, but the potential investment can be cut down if the SaaS application can reach the maturity Level-3, and the proposed SaaSify Tool could be well adopted.
- Required knowledge: For SOA approach, all the applications are considered as white boxes, they should be studied in detail. But for SaaS approach, the only one white box is the application which is chosen to be SaaSified. So the knowledge required by SOA approach is more than by SaaS approach.

After the comparison, the proposed SaaSified approach is superior to the traditional SOA approach in many areas such as required less knowledge and better scalability. What's more, the proposed SaaSified approach can avoid many problems such as data security, SLA and sustainability.

6 Conclusions

In this paper, an SaaS-based approach for EAI is proposed. It rebuilds one selected analogue of legacy applications in different branches, makes the rest of applications to be configured from the SaaSified one by customs, and further solves the EAI problems. The primary works of this research has following aspects:

- Propose an SaaS-based approach for EAI.
- Update the SaaS maturity model for the proposed approach.
- Explain how the SOA technique can be adopted to integrate SaaSified application.
- Propose a tool to introduce configuration capabilities to original web application automatically.

The proposed approach enhances applicability of SaaS paradigm and avoids shortcomings of traditional EAI approach. Our future work is to implement an SOA-based tool for integrating SaaSified applications by the approach which is proposed in Sect. 3.

Acknowledgments Jie Song, National Natural Science Foundation of China under Grant (61433008). Yichuan Zhang, National Natural Science Foundation of China under Grant (61502090). Beilei Wang, the School Basic Scientific Research Business Expenses for Northeastern University Grant (N130317004).

References

1. Messerschmitt DG, Szyperski C (2005) Software ecosystem: understanding an indispensable technology and industry. Springer, US
2. Hohpe G, Woolf B (2003) Enterprise integration patterns: designing, building, and deploying messaging solutions, 1st edn. Addison-Wesley Professional, Boston
3. Sushil J (2006) Enterprise flexibility. Glob J Flex Syst Manag 2:53–58
4. He X, Li H, Ding Q (2009) The SOA-based solution for distributed enterprise application integration. In: IFCSTA, pp 330–336
5. Ji X (2009) A web-based enterprise application integration solution. In: ICCSIT, pp 135–138

6. Chen M (2009) Research and implementation on enterprise application integration platform. In: IFITA, pp 93–96
7. Karabulut Y, Nassi I (2009) Secure enterprise services consumption for SaaS technology platforms. In: ICDE, pp 1749–1756
8. Liu F, Li L, Chou W (2009) Communications enablement of software-as-a-service (SaaS) applications. In: GLOBECOM, pp 1–8
9. Yan J, Zhang B (2009) Support multi-version applications in SaaS via progressive schema evolution. In: ICDE, pp 1717–1724
10. Lu Y, Sun B (2009) The fitness evaluation model of SAAS for enterprise information system. In: ICEBE, pp 507–511
11. Nilsson EG, Nordhagen EK, Oftedal G (1990) Aspects of systems integration. In: ICSI, pp 434–443
12. Daniel F, Jin Y, Benatallah B, Casati F, Matera M, Saint-Paul R (2007) Understanding UI integration: a survey of problems, technologies, and opportunities. IEEE Internet Comput (INTERNET) 11(3):59–66
13. Doan AH, Halevy AY (2005) Semantic integration research in the database community: a brief survey. AI Mag (AIM) 26(1):83–94
14. Studer R, Grimm S, Abecker A (2007) Semantic web services: concepts, technologies and applications. Springer, London
15. Paolucci M, Sycara K (2004) Ontologies in Agent Architectures. In: Handbook on Ontologies, 1st edn. Springer, New York, p 343–364
16. Chen H, Yin J, Jin L (2007) JTang synergy: a service oriented architecture for enterprise application integration. In: CSCWD, pp 502–507
17. Scheibler T, Mietzner R, Leymann F (2008) EAI as a service: combining the power of executable EAI patterns and SaaS. In: EDOC, pp 107–116
18. Vouk MA (2008) Cloud computing-issues, research and implementations. In: 30th International conference on information technology interfaces, pp 31–40
19. Gonzalez LMV, Rodero-Merino L, Caceres J, Lindner MA (2009) A break in the clouds: towards a cloud definition. Comput Commun Rev (CCR) 39(1):50–55
20. Fowler MJ (2003) Patterns of enterprise application architecture. Addison Wesley Professional, Boston
21. Laplante PA, Zhang J, Voas JM (2008) What's in a Name? Distinguishing between SaaS and SOA. IT Prof (ITPRO) 10(3):46–50
22. Wei Y, Brian Blake M (2010) Service-oriented computing and cloud computing: challenges and opportunities. IEEE Internet Comput (INTERNET) 14(6):72–75