Future Computing and Informatics Journal

Volume 1
Issue 1 (2016) *2016, Issues 1–2*

Article 3

2016

A Cloud Interoperability Broker (CIB) for data migration in SaaS

Hassan Ali
Arab Academy for Science, Technology & Maritime Transport (AASTMT), Cairo, Egypt,
hassanali83@gmail.com

ramdan mowad
ramdan.mowad@fue.edu.eg

amira Farouk
Arab Academy for Science, Technology & Maritime Transport (AASTMT), Cairo, Egypt,
amirahosni@link.net

Available online at www.sciencedirect.com

ScienceDirect

Future Computing and Informatics Journal 1 (2016) 27–34
http://www.journals.elsevier.com/future-computing-and-informatics-journal/

A Cloud Interoperability Broker (CIB) for data migration in SaaS

Hassan Ali [a], Ramadan Moawad [b,*], Amira Ahmed Farouk Hosni [a]

[a] Department of Postgraduate Computer Science Studies, College of Computing & Information Technology, Arab Academy for Science, Technology & Maritime Transport (AASTMT), Cairo, Egypt

[b] Department of Computer Science, Faculty of Computer and Information, Future University in Egypt, Cairo, Egypt

Received 16 June 2016; revised 16 February 2017; accepted 12 March 2017
Available online 16 March 2017

Abstract

Cloud computing is becoming increasingly popular. Information technology market leaders, e.g., Microsoft, Google, and Amazon, are extensively shifting toward cloud-based solutions. However, there is isolation in the cloud implementations provided by the cloud vendors. Limited interoperability can cause one user to adhere to a single cloud provider; thus, a required migration of an application or data from one cloud provider to another may necessitate a significant effort and/or full-cycle redevelopment to fit the new provider's standards and implementation. The ability to move from one cloud vendor to another would be a step toward advancing cloud computing interoperability and increasing customer trust. This study proposes a cloud broker solution to fill the interoperability gap between different software-as-a-service providers. The proposed cloud broker was implemented and tested on a real enterprise application dataset. The migration process was completed and it worked correctly, according to a specified mapping model.

Keywords: Cloud computing; Interoperability; SaaS broker

1. Introduction

The design of modern cloud computing technologies does not consider interoperability [1]. To facilitate cloud-computing interoperability, a previous work [2] proposed that every cloud system be described by its components, including resources, services, and Application Program Interfaces (APIs). Moreover, widely accepted standardized APIs should be used to regulate communication and resource and services management. However, such a model is not realistic and would be difficult to achieve because companies consider differentiation a competitive advantage. Consequently, another approach using a cloud broker was introduced [2]. A cloud broker is used as a mediation technique to enable interaction between different cloud systems that were not equipped with the ability to interact with each other. This technique may reduce the need for modifying current cloud systems. Finding solutions to the issues related to interoperability could advance cloud computing because this will increase customer trust in the cloud and solve vendor lock-in problems. According to "Cloud Computing Use Case Discussion Group, Cloud Computing Use Cases V.4," there are two aspects of cloud computing interoperability, namely, technical (syntactic) and semantic. Technical interoperability is the ability of a code to simultaneously work with multiple cloud vendors. It includes aspects such as interface specifications, services for interaction, and integration, presentation, and exchange. "European Interoperability Framework (EIF) for European Public Services" describes semantic interoperability as the ability of an organization to understand information coming from external

* Corresponding author.
E-mail addresses: hassanali83@gmail.com (H. Ali), ramdan.mowad@fue.edu.eg (R. Moawad), amirahosni@link.net (A.A.F. Hosni).

Peer review under responsibility of Faculty of Computers and Information Technology, Future University in Egypt.

http://dx.doi.org/10.1016/j.fcij.2017.03.001

sources with the same meaning as the originator intended. In this paper, we introduce a Cloud Interoperability Broker (CIB) at the Software-as-a-Service (SaaS) level in an attempt to bridge the semantic interoperability gap between different SaaS providers that serve the same business need but still lack interoperability prospects as a result of not following the same data model. Although the idea of filling the interoperability gap is not new, most efforts have addressed its challenges in the Infrastructure-as-a-Service (IaaS) layer. For example, the RESERVOIR model is aimed at creating a service model for providing and managing resources and services transparently on an on-demand basis. The RESERVOIR model aims to enable IaaS vendors to dynamically interoperate to create what could be considered as a pool of infinite IT resources [1]. In Ref. [3] the authors use the concept of brokering to optimize virtual infrastructure deployment via multiple cloud systems, regardless of the deployment and management of physical infrastructures. The novel idea introduced in this paper is to apply the brokering concept in the SaaS layer. The remainder of the paper is organized as follows. Section 2 provides a survey of existing cloud-computing interoperability efforts. Section 3 introduces the proposed cloud broker methodology. Section 4 presents the cloud broker's implementation. Section 5 summarizes the paper and discusses future work.

2. Related work

Several studies have addressed cloud-computing interoperability challenges. Some have introduced a mediation mechanism, while others have suggested standards to be adopted by any cloud provider for interacting seamlessly with others. The research efforts are summarized below.

2.1. Extended levels of Information System Interoperability (LISI) model [4]

The extended version of the LISI Maturity Model was proposed by the Department of Defense (DoD) C4ISR Architecture Working Group [4]. They assumed that the first two levels of the original LISI model are a must for any enterprise intending to relate to the cloud. They also assumed that those enterprises have sufficient security maturity to reach the third level in the original model and then added another three additional levels. The new extended model proposed the following five levels of cloud-to-cloud interoperability:

- Level 1: Domain-based interoperability. This level provides for an integrated environment that has the following features: a Wide Area Network (WAN), data, and databases are shared among applications that are separated or isolated from each other. In addition, cloud services are restricted to a single cloud vendor.
- Level 2: Enterprise-based interoperability. This level provides for a universal environment that has a WAN, shared data, shared applications, and a capability for sharing information across domains. The cloud services can be used for advanced collaboration between different cloud systems.
- Level 3: Portability interoperability. This level provides for different cloud environments in which cloud artifacts such as virtual machines can be moved between multiple providers in down states. Security, privacy policies, and Service-Level Agreement (SLA) enforcement are based on a common agreement between providers.
- Level 4: Security interoperability. This level provides for different cloud environments in which policies and procedures can interact transparently and automatically with their equivalents across different vendors using standard protocols, with a unified security model that is approved by all of the vendors.
- Level 5: Mobile Interoperability. This level provides for different cloud environments in which cloud artifacts can be moved between multiple vendors in running states. Data and applications are interrelated.

The first three levels are attainable and can be achieved through the use of interoperability brokers, virtual machines (VMs), interoperability formats, such as open virtualization format (OVF) or the extended version mentioned in Ref. [5]. Standard protocols, formal trust relationships, new innovative technologies, and cultural transformations in enterprises thinking to gain acceptance for the idea of sharing infrastructure and/or applications are needed to support levels four and five.

2.2. National Institute of Standards and Technology (NIST)

NIST is an organization that works with enterprises to create and apply standards and measurements for technology. It developed the following five use cases for cloud computing interoperability concerning the manner of interaction between cloud consumers and vendors:

- Copy data objects between cloud-providers: This involves copying data objects from one cloud provider system to another.
- Dynamic operation dispatch to IaaS clouds: This involves invoking operations on the most effective available clouds, based on a set of rules identified on the client side. Those rules are evaluated at runtime.
- Cloud burst from data center to cloud: This involves dynamically allocating or de-allocating cloud computing processing capabilities or storage resources to maintain required service level agreements for a company's data center-hosted processes.
- Migrate a queuing-based application: This involves migrating an existing queue with its messages from one cloud provider to another.
- Migrate (fully stopped) VMs from one cloud provider to another: This involves seamlessly migrating a fully stopped virtual machine from one cloud provider to another.

Our work relates to the first and fourth use cases for migrating data entities and workflows, respectively, between different SaaS providers.

2.3. Object Management Group (OMG)

OMG works to create standards for integrating a variety of technologies for many industries. It conducted a meeting for cloud vendors and consumers to agree upon a set of use cases for cloud computing in more generic terms than those developed by NIST. The use case closest to our work is the "Changing Cloud Vendors" use case, especially the first scenario, because it is concerned with changing from one vendor to another.

Scenario1: Changing SaaS vendors: This scenario involves changing from one SaaS vendor to another. Source and destination applications serve the same business domain (e.g., Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), or structured or unstructured data). The data and documents created with one vendor can be imported to the second vendor's applications.

"Scenario requirements": Standards state that vendors' applications must support common formats for data and document migration from one provider to another, with those formats depending primarily on the type of application. Standard APIs are sometimes required for different application types. We need to know that those requirements are not specialized for individual cloud application, for example, the standards used to move a document from Zoho Docs to Google Docs are the same as those used for migrating one from Microsoft Office to OpenOffice.

2.4. Role of standards in cloud-computing interoperability

This paper explores areas of cloud computing in which standardization would benefit interoperability, and other areas in which would not [6]. The author extracted four use cases regarding consumer–provider interactions from other standards. The third one on data migration relates to cloud computing interoperability.

"Data Migration use case" This involves enabling data located in one cloud vendor to be migrated to another one. The author mentions that data to be moved must be extracted from the source SaaS provider and uploaded to the destination. He also cites some existing standards that support the goal of data movement, such as the cloud data management interface (CDMI), Simple Object Access Protocol (SOAP), and Representational State Transfer (REST) APIs. We propose a CIB that acts as a middleware layer whose principal function is to move data from one SaaS provider to another. Data to be extracted from the source are normalized to match the destination data structure and then uploaded to the destination. The primary goal of this work is to test the applicability of the data migration process from one cloud vendor to another and test its results.

3. SaaS CIB methodology

This study aims to create a CIB in the SaaS layer that would act as a mediator for filling in the interoperability gap between any SaaS providers that serves the same business need or domain. Many research efforts have addressed the idea of brokerage but the implementation and testing of a mediation mechanism in the SaaS layer is a novel idea. In addition, this research focuses on developing a generic cloud interoperability brokerage methodology that can be implemented for the different data migration and movement processes in the SaaS layer. Fig. 1 shows the location of the CIB in the SaaS layer, and more generally, in relation to the cloud computing deployment models IaaS, SaaS, and Platform as a Service (PaaS).

The CIB serves as a mediator between the different types of applications hosted in the SaaS layer that serve the same business need. CIB uses the standard APIs exposed by each SaaS provider to extract data and then transforms it into a form that can be provided to the other SaaS providers.

We divide the data into two principal categories, namely, master data and transaction data as follows:

Master data: It refers to the core objects in the business application, such as suppliers, customers, products, employees, and assets.

Transaction data: It refers to application transactions reflecting changes in application object states. These data are created in the context of business processes and they reference master data and reflect changes in them. The relationships among master and transaction data are similar to a parent–child relationship. Transactional data related to sales, purchases, activities and cancellations reference master data, which are the principal objects of business.

The proposed methodology involves the following five steps:

3.1. Collection and analysis of the metadata

We need to determine the data tables and their attributes. In addition, we need to know their meanings. Each attribute must be identified by name, data type, relations, constraints, and default values.

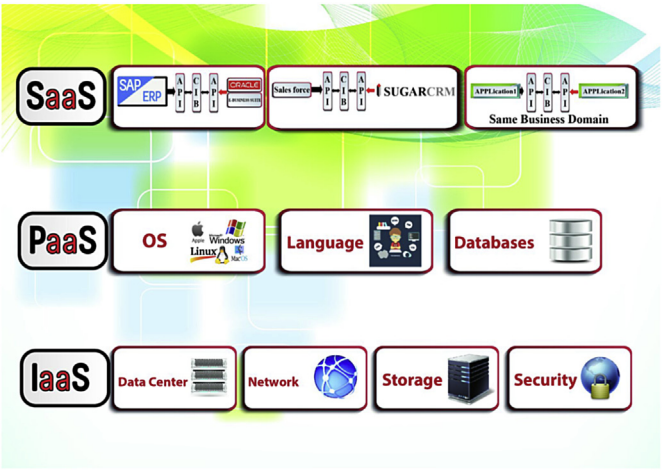

Fig. 1. CIB location inside SaaS layer of cloud computing layers.

3.2. Develop the mapping model

The core function of the mapping model is to create a relation between each entity and its equivalent on SaaS application to be mapped to. This is achieved by deciding how the records look like, which attributes are included, of which size and data type they are, and which values are allowed. This step should also include the mapping between the source and the destination entities' attributes. This is normally both the most important and the most difficult step in the process.

3.3. Solution design

Once the mapping process is completed, the solution design must be created as a software solution that can be implemented to accomplish the data migration goal.

3.4. Implementation

Implement the previous solution design as software that abstracts each SaaS provider's APIs and preforms the data migration.

3.5. Test the solution

This step is the most challenging. It requires an iterative effort and manual inspection to ensure the completeness and accuracy of the data movement.

A simple flowchart presenting this methodology is shown in Fig. 2.

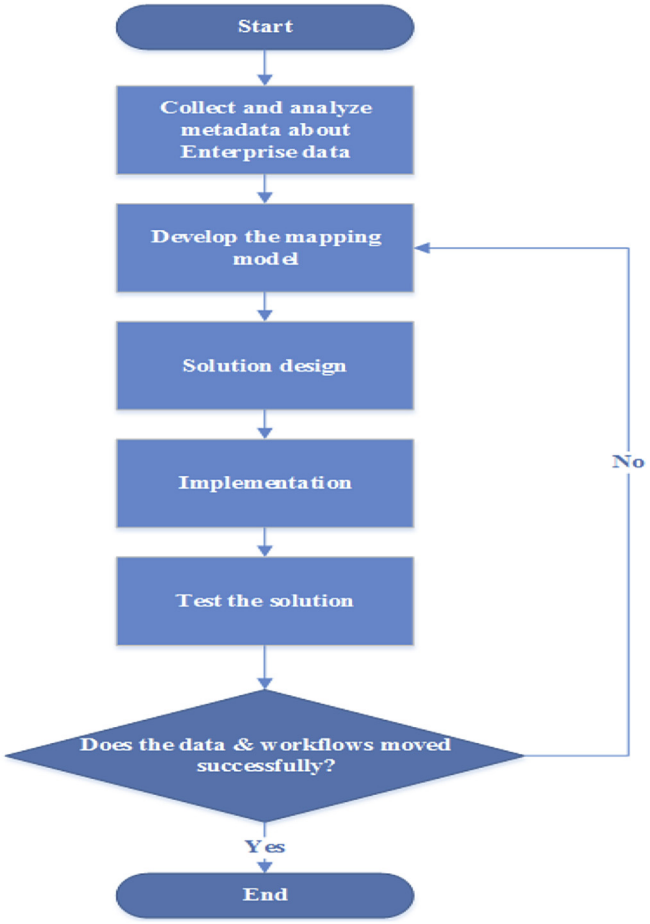

Fig. 2. SaaS CIB methodology steps.

4. CIB implementation (case study)

The case study is an application of the proposed methodology on two well-known SaaS providers, namely, Microsoft (MS) Dynamics CRM 2015 and SugarCRM. However, this methodology can be used for other SaaS providers, e.g., SAP ERP, and Oracle E-Business suites ERP applications. CRM is an approach to manage a company's interactions with customers, vendors, or prospects using one or more touching points [7]. It often involves using technology to organize, automate, and synchronize sales, marketing, customer service, and technical support. Those are the principal modules or components that must be included in any CRM application. Commonly used master data entities that are shared across all of the CRM Modules include the following entities: Account, Contact, Lead, System User, Product, and Contract. Transactional entities may include Opportunity, Activities, Team, Calendar, Quote, Order, Invoice, Document, Email, Knowledge Base Article, and Service Appointment.

Those entities' names might vary from one application or service provider to another but they must exist in some form, and some additional entities might also exist specifically for one provider. The solution in this case is to use custom entities to apply the mapping if they do not exist on the destination application.

Applying the methodology:

4.1. Collecting and analyzing metadata

Each entity has been described and mapped to its equivalent business unit. The relation between business unit and data entity could be from one to one or one to many; for example, the business unit Contact in MS Dynamics CRM 2015 is mapped to the ContactBase, ContactInvoices, ContactLeads, ContactOrders, and ContactQuotes data entities. Then, each business entity is described in the context of its data type, allowed values, constraints, default values, and dependencies. An example of entity data mapping is described in Table 1.

4.2. Developing a mapping model for attributes

The mapping of attributes and data types for the example entity between the two SaaS providers would be as in Table 1. We must also note that the mapping table or model for each of the other entities would be performed in the same way, but the attributes' names and data type would be different. Consequently, we do not need to show the other tables in this paper.

4.3. Solution design

1) *Design Considerations and highlights:* Before going on with the design details, it is essential to highlight some

Table 1
Mapping Table for some attributes of ContactBase Entity between the Microsoft & SugarCRM SaaS Providers.

Microsoft (MS) Dynamics CRM Contactbase Entity	MS Data Type	SugarCRM Contacts entity	SugarCRM Data Type
ContactId	uniqueidentifier	Id	uniqueidentifier
CustomerTypeCode	Int	CustomerTypeCode (custom attribute lockup)	Int
PreferredContactMethodCode	Int	PreferredContactMethodCode (custom attribute lockup)	Int
LeadSourceCode	Int	lead_source	string
ShippingMethodCode	Int	ShippingMethodCode (Custom Attribute Lockup)	int
Salutation	String	Salutation	String
JobTitle	String	Title	String
FirstName	String	first_name	String
NickName	String	Nickname Custom Attribute	String
MiddleName	String	MiddleName Custom Attribute	String
LastName	String	Last_Name	String
Suffix	String	Suffix	String
BirthDate	Date	BirthDate	Date
Description	String	Description	String
GenderCode	int	Gender	String
AnnualIncome	Money	AnnualIncome Custom Attribute	Money
WebSiteUrl	String	portal_name	String
EMailAddress1	String	Email1	String
EMailAddress2	String	Email2	String
DoNotPhone	Bit	DoNotPhone (custom Attribute)	Boolean
CreatedBy	Uniqueidentifier	created_by	Uniqueidentifier
ModifiedOn	Date	Modified_date	Date
ModifiedBy	Uniqueidentifier	Modified_By	Uniqueidentifier
Telephone1	String	phone_home	String
Telephone2	String	phone_other	String
Telephone3	String	phone_work	String
Fax	String	phone_fax	String
TransactionCurrencyId	uniqueidentifier	Currency Custom Attribute (lockup)	Uniqueidentifier

design considerations regarding the proposed methodology in the form of a broker:

a) *Selection of a standard:* The broker consumes the available API's of the two SaaS providers in the form of XML Web services. It exchanges messages with applications through those services using the SOAP messaging format.

b) *Technical decisions and workarounds:* While the project is being implemented, some technical decisions and workarounds must be made in order to ensure that the work is performed. For example,

when the SugarCRM interface is being used to create an Account record, only the mandatory attributes (name, phone, and website) are used. We can not add extra attributes other than those in the create_account function. That leads to a technical decision to create the record using those base attributes and then saving its ID (the primary key that uniquely identifies the record) to a locally created database to keep a cross-reference between primary keys of newly created records and the originals. This ID is then used to update the created record with the remaining entity attributes.

There is no function to create or manage workflows in SugarCRM. This has been overcome by using functions that create records directly in database tables to create workflow, actions, and triggers directly.

Finally the classes are not strongly typed, with the result that a mistyped entity or attribute name will not be validated until runtime. That is not the case when working with the Microsoft API.

c) *User Interaction:* CIB has a separate set of classes for each provider to deal with its principle functions and with any other function required to work with this entity. For some entities, there are extra attributes that have no equivalent on the other application. Those attributes' types range from simple data types to complex tables. This problem can be addressed by creating custom fields on the opposite application. In MS Dynamics CRM we can create those custom fields through code using APIs, but in SugarCRM this facility is not available through APIs. The user is required to create them manually through an application's own graphical user interface (GUI). What CIB can do is to warn the user regarding the missing attributes and the need to create them.

2) Design Steps:

a) *Connection Handling:* The connection to the providers' APIs is handled through a class that is referenced in the primary interface containing the signature of common behaviors for all entity wrapper classes. Each class must implement that interface.

b) *Entity Handling Classes:* Each entity is wrapped up through two classes, one for business logic and the other for data attributes. The data attributes class is intended for isolating entities from naming differences in the respective SaaS providers, and the caller function is responsible for passing suitable attributes according to the source and destination. For example, the fax attribute in the CIB "UserData" class has the name phone_fax in the SugarCRM SaaS Provider "Users" entity, and address1_fax in Microsoft CRM Provider. A typical example of an entity classes diagram is shown in Fig. 3.

c) *Data Movement Process:* The movement of data passes through a three principal data mining steps: Extract, Transform, and Load (ETL).

- Extract: this is the step for extracting data from the source application.
- Transform: This step can contain any preparation of data to match the destination application's acceptable formats. For example, the field that shows that the entity is disabled in Dynamics CRM is a Boolean (contains whether true or false), while in SugarCRM it is a string value (Active or Inactive); hence, the transform process might include this step for each entity that has disabled or deleted flags.
- Load: This is the process of uploading data to the destination application. In fact, this is a two-step process, because we are first moving the basic data (the date that do not reference other data) and then another step is required to set up references. this step includes using a locally created database that stores the new values of the primary key attribute of a parent record in the destination provider along with the original value in the source provider. After the movement process is completed a function is used to update the foreign keys that reference this primary key with its new value, thus the relationship between entities is migrated to the destination provider to guarantee the consistency of related entities. In addition, another optional step is required when the destination is SugarCRM, as the creation functions are limited to mandatory attributes only, in this case the update reference function contains another step to fill in the rest of data as well as setting references or using a separate function.

4.4. Implementation

The implementation of the previous design for our broker is written in C# programming language within the Microsoft Visual Studio 2012 integrated development environment.

4.5. Test the solution

We use sample customer relationship management data for an air conditioner and television monitor manufacturing and sales company with approximately ten thousand records for different application modules. The main testing criterion in this phase of our project is the completeness of the data movement process from source to destination and the

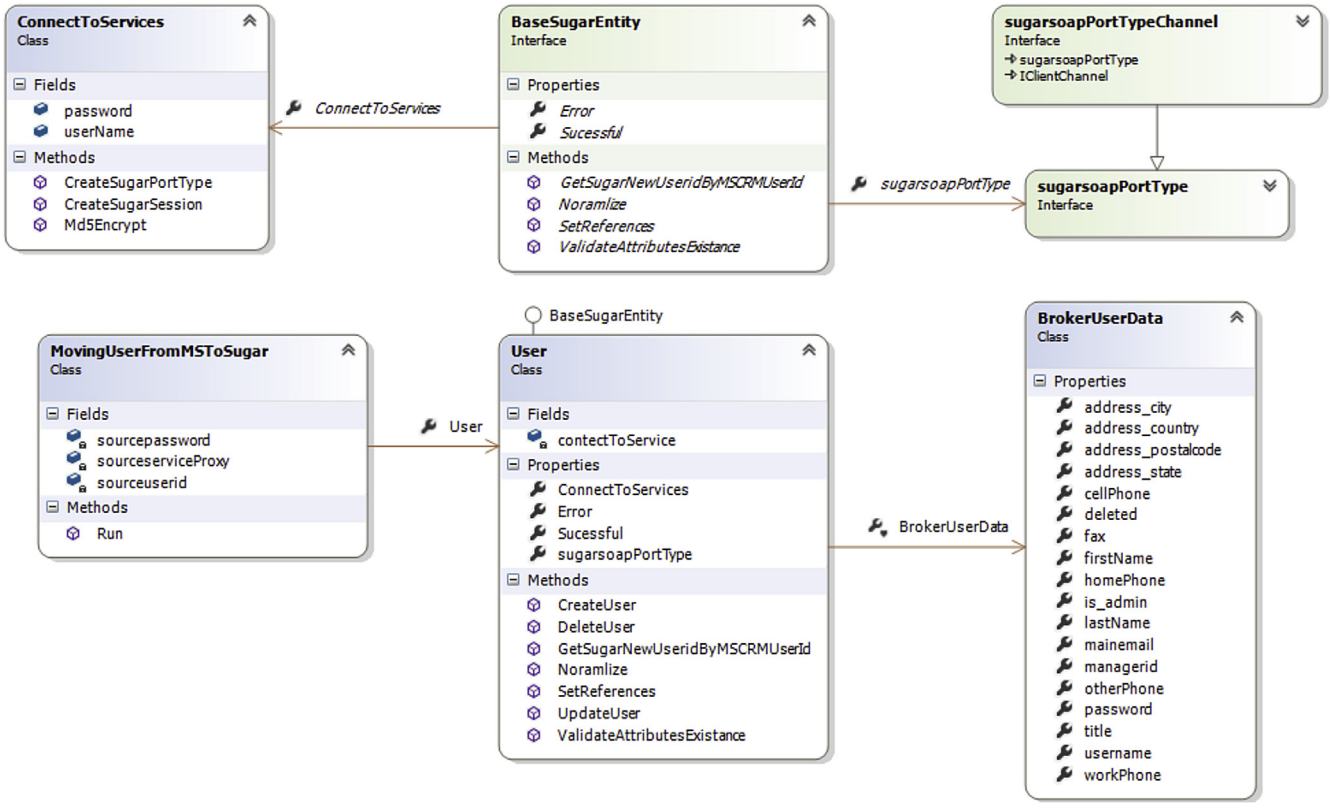

Fig. 3. User entity handling classes class diagram.

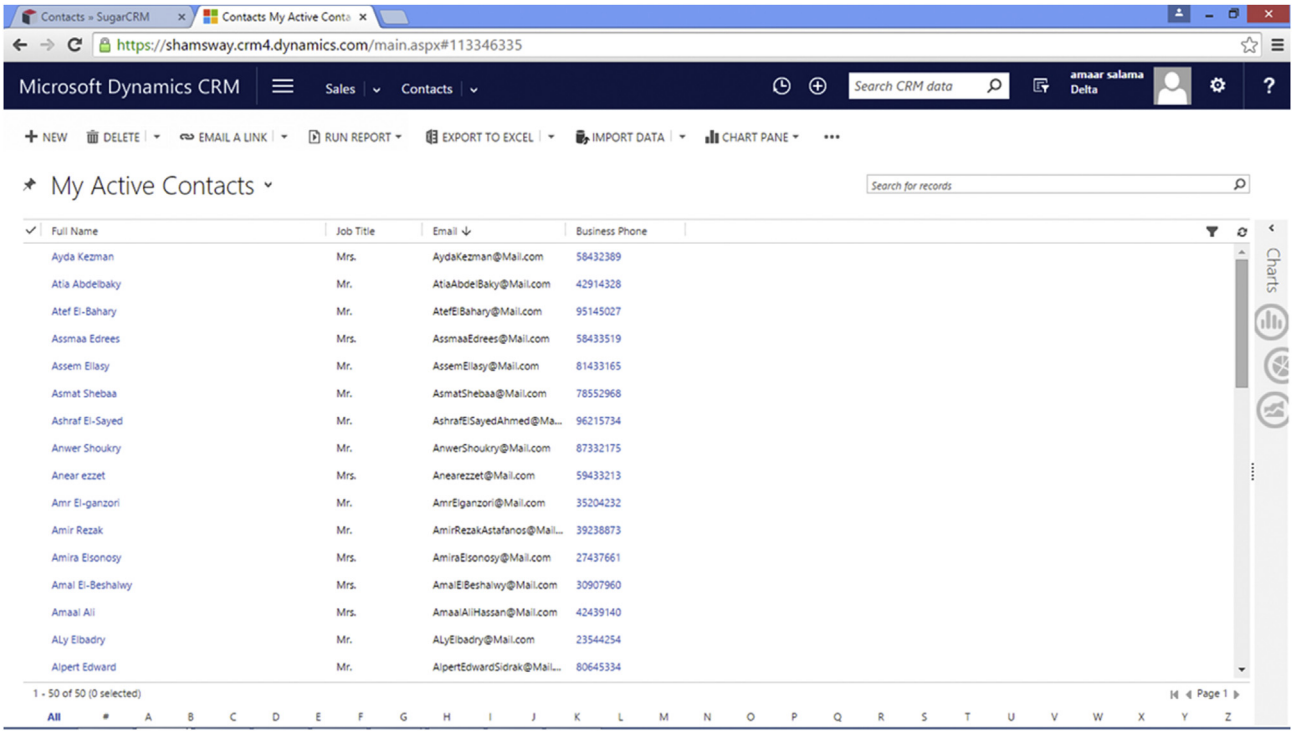

Fig. 4. List of contacts in MS Dynamics CRM 2015 SaaS provider.

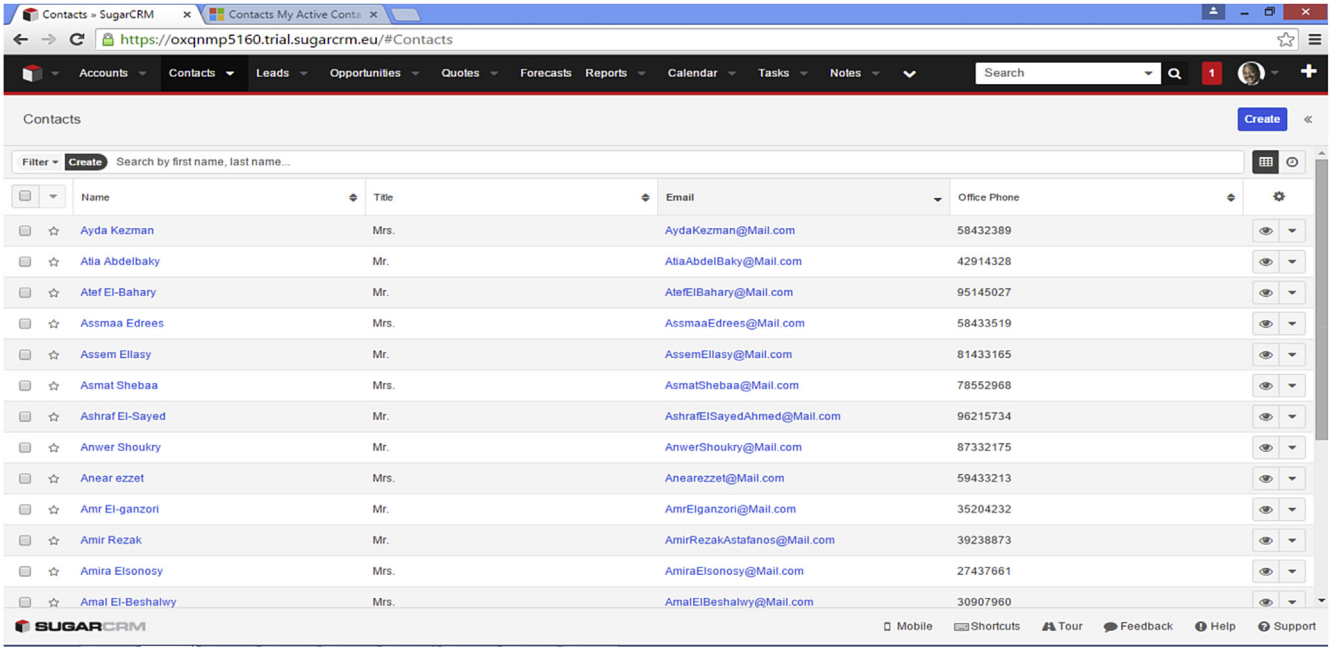

Fig. 5. List of contacts in SugarCRM SaaS provider after being moved using CIB.

correctness of these data after being moved. Figs. 4 and 5 show a list of contacts entities in the source (MS Dynamics CRM 2015 SaaS provider) and their copy of that data using our broker, to the destination (SugarCRM SaaS provider). As shown in the figures, the data has been copied successfully from source to destination and were mapped correctly according to the mapping model for those entities.

5. Conclusion and future work

In this research, paper a methodology was developed for a CIB at the SaaS layer of cloud computing. The methodology was then implemented in a case study to evaluate and initially test the broker on two SaaS providers as a proof of concept of the application. The desired results were obtained, meaning

that, mapping and migrating data from one provider to another and vice versa to overcome the vendor lock-in problem in the SaaS layer of cloud computing paradigm are possible and can be achieved using the proposed CIB.

A future work would include the following:

- Provide a methodology and guide lines for creating APIs for SaaS providers that serve the goal of interoperability, and implement more use cases to demonstrate the flexibility of the proposed methodology.
- Provide a user interface with initial mapping of entities' attributes that can be modified by the user and then apply data movement according to that mapping.
- Conduct more analysis of data movement process using different evaluation criteria.
- Evaluate the broker in relation to different aspects, such as performance, accuracy, time.

References

[1] Rochwerger B, Breitgand D, Levy E, Galis A, Nagin K, Llorente IM, et al. The RESERVOIR model and architecture for open federated cloud computing. IBM J Res Dev 2009;53(4):535–45.

[2] Loutas N, Kamateri E, Bosi F, Tarabanis K. Cloud computing interoperability: the state of play. In: Cloud computing technology and science (CloudCom), Athens, Greece. IEEE; 29 November – 1 December, 2011. p. 752–7. IEEE Third International Conference.

[3] Tordsson Johan, Montero Rubén S, Moreno-Vozmediano Rafael, Llorente Ignacio M. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. Future Gener Comp Sy 2012;28(2):358–67.

[4] Dowell Scott, Barreto Albert, Michael Bret, Shing Man-Tak. Cloud to cloud interoperability. In: System of systems engineering, Albuquerque, New Mexico, USA. IEEE; 27–30 June, 2011. p. 258–63. Proceedings of the 2011 6th International Conference on System of Systems Engineering.

[5] Maximilien E, Ranabahu Ajith, Engehausen Roy, Anderson Laura C. Toward cloud-agnostic middlewares. In: Object oriented programming systems Languages and applications (OOPSLA 2009), Orlando, Florida, USA. ACM; 2009. p. 619–26. Proceeding of the 24th ACM SIGPLAN conference.

[6] Lewis GA. Role of standards in cloud-computing interoperability. In: System sciences (HICSS), Wailea, Maui, Hawaii, USA. IEEE; 7–10 Jan. 2013. p. 1652–61. 46th Hawaii International Conference on System Sciences.

[7] Liu ChunNian, Wang YongLong, Pan Qin. Construction and development of CRM technology and industry chain in China. In: Intelligent computing and information science. Berlin, Germany: Springer Berlin Heidelberg; 2011. p. 118–23.