ENHANCING CLOUD SAAS DEVELOPMENT WITH MODEL DRIVEN ARCHITECTURE

Ritu Sharma[1] and Manu Sood[2]

[1, 2] Himachal Pradesh University Summer Hill, Shimla 5,
Himachal Pradesh, India
[1]rituchetan@gmail.com, [2]soodm_67@yahoo.com

ABSTRACT

Cloud computing is a leading edge computing paradigm in which massively scalable and dynamically configured resources are offered remotely as services, over a network. With rapid transitions in hardware and software technologies, as witnessed in the recent years, the biggest challenge posed to the IT industry is technology obsolescence. Consequently, there arises a need for a software development approach that could mitigate the undesirable effects of technology change. The Model Driven Architecture (MDA) approach to software development becomes an obvious choice. In this approach, the models drive the process of software development. These models are specified at different levels of abstraction and automated tools are used for model-to-model and model-to-code transformations between the levels. In this paper, the authors endeavour to explore the MDA approach for developing cloud software applications so that the ensuing software solutions are more robust, flexible and agile, in the wake of constantly evolving technologies.

KEYWORDS

Cloud computing, Cloud Software-as-a-Service (SaaS), Model-Driven Architecture (MDA), Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM), Model Transformation

1. INTRODUCTION

Cloud computing is a recent trend in the field of Information and Communication Technology (ICT), wherein computational resources like hardware, software, development environment and other infrastructure are provided as services, remotely over a network (intranet or Internet). These services can be easily accessed across a simple interface, such as a browser, running on a thin client or even a mobile phone. The cloud services are analogous to other utility services such as electricity, water etc as they are made available to the consumers on-demand on a pay-per-use or subscription basis. The cloud maintains a shared resource pool and the resources are acquired from or released to this pool dynamically to meet the varying demands of the consumer thereby giving him an illusion of existence of infinite resources.

The rapid advancements observed in the field of ICT require software solutions to be developed in a manner that is independent of technology change. Model Driven Architecture (MDA) is a software development approach where the models are used as prime artefacts throughout the process of software development. These models are defined at different levels of abstraction to represent various aspects of the system. Besides, they are formal in nature and can be machine processed. The transformation of models from one level of abstraction to another, or the

DOI : 10.5121/ijccsa.2011.1307

transformation of models to executable code is performed by using (semi)automated transformation tools.

This paper attempts to leverage the MDA approach in the development of cloud SaaS (Cloud Software-as-a-Service). A platform-independent model (PIM) of the cloud SaaS would reflect its structure, behaviour and functionality irrespective of the technology used for its implementation. The platform-specific model (PSM) of the cloud SaaS, on the other hand, would be more implementation-oriented and bound to a given execution platform. The transformations from the PIM to PSM would be carried out using transformation tools developed for the purpose. Section 2 and 3 discuss the fundamental concepts of cloud computing and MDA, respectively. Section 4 discusses the abstraction layers of a Cloud SaaS with respect to MDA. Section 5 illustrates a cloud SaaS, the Online Hotel Reservation System (OHRS), which has been taken as an example for demonstrating the model-driven software development approach. The subsections in Section 5 describe the CIM, PIM and PSM of a cloud software application taken as example and also discuss the transformation rules for transforming a PIM into two PSMs – Relational PSM and EJB PSM. Section 6 draws the conclusion of the paper and the future work undertaken by the authors.

2. CLOUD COMPUTING

Cloud computing represents the fifth generation of computing after mainframe computing, personal computing, client-server computing and web computing [1]. It is a modern approach to build and remotely manage computing resources. The user is only required to establish an account with the service provider in order to utilize a service. Although the term cloud was first used in early '90s to refer to large ATM networks, cloud computing became available commercially at the beginning of this century with the advent of Amazon's Web Services. The other big vendors who gradually joined the fray include Yahoo, Google, Microsoft, IBM, Sun, Intel, Oracle and Adobe. A cloud, in the given context, refers to a complex, internet-based infrastructure of hardware and software components [2, 3].

Cloud computing has evolved from a range of relevant legacy technologies and concepts such as grid computing, virtualization, Web Services and its supporting technologies (such as Web Service Definition Language (WSDL), Simple Object Access Protocol (SOAP) and Universal Description Discovery and Integration (UDDI)), Service Oriented Architecture (SOA), Software as a Service (SaaS) etc.

Cloud computing encapsulates a variety of services such as – Software-as-a-Service (SaaS) wherein the software applications running on a cloud infrastructure are offered as services to the consumers; Platform-as-a-Service (PaaS) in which development platforms and technologies are delivered as services to the consumers; and Infrastructure-as-a Service (IaaS) where processing, network, storage and other fundamental computing resources are provided for deploying and running the software. In the absence of a standard taxonomy, other suggested service categories include Hardware-as-a-Service (HaaS), Development, Database and Desktop as a Service (DaaS), Business as a Service (BaaS), Framework as a Service (FaaS), Organization as a Service (OaaS) etc [4, 5, 6].

A cloud may be deployed as a private cloud, public cloud or a hybrid cloud. The cloud infrastructure in a private cloud operates solely for an organization, while in a public cloud the infrastructure is available to general public. A hybrid cloud is composed of two or more clouds of different types such as public or private.

The primary force that drives cloud computing is the economies of scale for both – the service providers and the service consumers. From a client's perspective, the cloud services are faster, simpler and cheaper to use and can be accessed from anywhere at any time. Also, it does not require any upfront infrastructure costs and operational expenses on part of the customer. From the provider's perspective, cloud computing facilitates cost-effective delivery of services and optimal utilization of shared and pooled resources. But, there are certain issues that need to be addressed before migrating to a cloud. These include security and privacy concerns for data in the cloud, regulatory issues, reliability of service in case of outage/failover, Service Level Agreements between the provider and consumer, latency period for real-time applications, vendor lock-in etc.

3. MODEL-DRIVEN ARCHITECTURE (MDA)

Model Driven Architecture (MDA), an initiative by Object Management Group (OMG), is an open, vendor-neutral approach to software development which is characterized by the use of models as primary artifacts for understanding, design, construction, deployment, operation, maintenance and modification of a system [7, 8]. It reflects separation of concerns by separating business functionality from implementation technology. While traditional software design and development processes create applications for deployment to a specific technology platform, MDA introduces higher levels of abstraction, enabling organizations to create models that are independent of any particular technology platform. The strength of MDA lies in the fact that it is based on widely used industry standards for visualizing, storing and exchanging software designs and models.

The models in MDA are abstracted at three different levels – the Computation Independent Model (CIM), the Platform Independent Model (PIM) and the Platform Specific Model (PSM). A CIM is a software independent business domain model that bridges the gap between business experts and system experts. A PIM specifies the functionality of the system independent of the technology that would be used for its implementation. A PSM specifies the system in terms of implementation constructs that are specific to the implementation technology. A single PIM can be transformed into one or more PSMs, each PSM being specific to the technology platform on which the system would finally be implemented.

A complete MDA specification consists of a definitive platform-independent model (PIM), plus one or more platform specific models (PSM), sets of interface definitions, each describing how the base model is implemented on a different middleware platform and sets of transformation definitions. The PIM depicting the structure, behaviour and functionality is modelled only once. And then, the transformation definitions enable the transformation of the PIM to one or more PSMs.

The key to the success of MDA lies in automated or semi-automated model-to-model and model-to-code transformations. The transformation tool executes a transformation definition that is specified for the purpose of transforming higher-level, platform-independent business models into lower-level platform-specific models and finally into executable code. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language [9]. Figure 1 depicts a PIM to PSM transformation.

Figure 1. PIM to PSM Transformation [14]

4. ABSTRACTION LAYERS FOR A CLOUD SAAS

According to US National Institute of Standards and Technology [4], a Cloud SaaS is a service model whereby the provider's applications running on a cloud infrastructure are offered to the consumer as a *service*. These applications can be accessed using a variety of devices through a thin client interface such as a browser. The consumer is not required to manage or control the underlying infrastructure, except for limited user-specific application configuration settings, if any.

With the hardware and software technologies in a state of flux, it is more appropriate to *model* the cloud software applications at a higher level of abstraction, instead of developing them directly using available technologies. This would decouple the software applications from the undesired effects of technology change and enhance their longevity. An MDA-based development of Cloud SaaS (application) would facilitate defining these services in a technology-independent manner and would play a significant role in improving their quality making them more robust, flexible and agile [10, 11]. Besides, encapsulating business logic independent of the technical mechanisms would formally capture the essence of the applications; and would also make it possible to reuse them in a variety of contexts [12, 13].

As mentioned earlier, the MDA approach would define the cloud software application at three different levels of abstraction. The CIM of the Cloud SaaS would specify the computation independent view of the system. It would capture the requirements of the system in a vocabulary familiar to the domain practitioners. The PIM of the Cloud SaaS would specify the system at the next lower level of abstraction. It would capture a platform independent view focusing on the operation of the system while hiding the platform specific details. The PSM of the Cloud SaaS would be defined at the next lower level of abstraction, focusing on the details of use of a particular platform, thereby providing a platform specific view of the system. Based on MDA, a single PIM for a cloud software application can be mapped to several different PSMs targeted on different platforms by defining platform-specific transformation rules.

5. AN ILLUSTRATION OF A CLOUD SAAS – ONLINE HOTEL RESERVATION SYSTEM (OHRS)

The model-driven approach to a cloud SaaS (application) development is illustrated with the help of an example – the Online Hotel Reservation System (OHRS) [14]. The OHRS is a software application that runs as a service in the cloud, performing a variety of tasks for its clients such as determining the availability of rooms in the hotel, online reservations, online cancellations, generating arrival chart, generating reports for decision-making, modifying room tariffs etc.

Besides, as the application is web-based, it can be accessed by the prospective customers of the hotel enterprise to determine the availability of accommodation, or to book accommodation, or to cancel a previously booked accommodation. The OHRS cloud service may thus be utilized by any small or medium scale enterprise (SME), on a payment basis. The proposal would also be beneficial for budding entrepreneurs who wish to start up with a hotel business on a small scale but do not wish to initially invest huge capital in purchasing, installing or developing the hardware and software for the purpose.

The CIM, PIM and PSM for the OHRS cloud application are discussed in the following subsections. Although, the MDA approach does not restrict itself to the use of UML (Unified Modelling Language) for modelling software applications, the authors are using UML to model various aspects of the system.

5.1. Computation Independent Model (CIM) for OHRS

As part of the CIM, the Use Case diagram and the Activity diagram available in UML have been used to model the functional requirements of the system under consideration. The loosely defined requirements of the business process are distilled into these requirements models without losing essential details.

A Use Case diagram for OHRS cloud application is depicted in Figure 2. A use case, represented by an oval in the diagram, captures a piece of functionality that the system provides. It specifies 'what' a system is supposed to do. The characteristics of the actors interacting with the OHRS are [14]:

- A Customer is a person who accesses the OHRS to view the availability status or to book/cancel an accommodation in the hotel
- An Administrator is the hotel personnel who accesses the OHRS for not only viewing the availability status or booking/cancelling accommodations for its customers over the counter, but also to generate various reports for decision making or for modifying the tariffs of unit types.

In OHRS, the process of booking an accommodation involves checking the availability and if available, verifying the payment details prior to booking. The Use Case diagram depicts this by means of `include` relationship which declares that the use case at the head of the dotted arrow completely reuses all the steps from the use case being included. The 'Book accommodation' use case includes the 'Check Availability' and 'Verify Payment Details' use cases. Similarly, the 'Cancel booking' use case includes the 'Verify Customer Information' use case. The 'View Customer Status' use case is a generalization of 'View Reservation' and 'View Cancellation' use cases. The `generalization` relationship represents use case *inheritance* by applying a use case with small changes for a collection of specific situations. Also, the 'Generate Report' use case is a generalization of 'Reservation Status', 'Cancellation Status' and 'Revenue Earned' use cases. The 'Cancel booking' use case extends 'Generate Cancellation Receipt' use case. An `extend` relationship represents an *optional* reuse of a use case depending on a runtime behaviour or a system implementation decision [15].

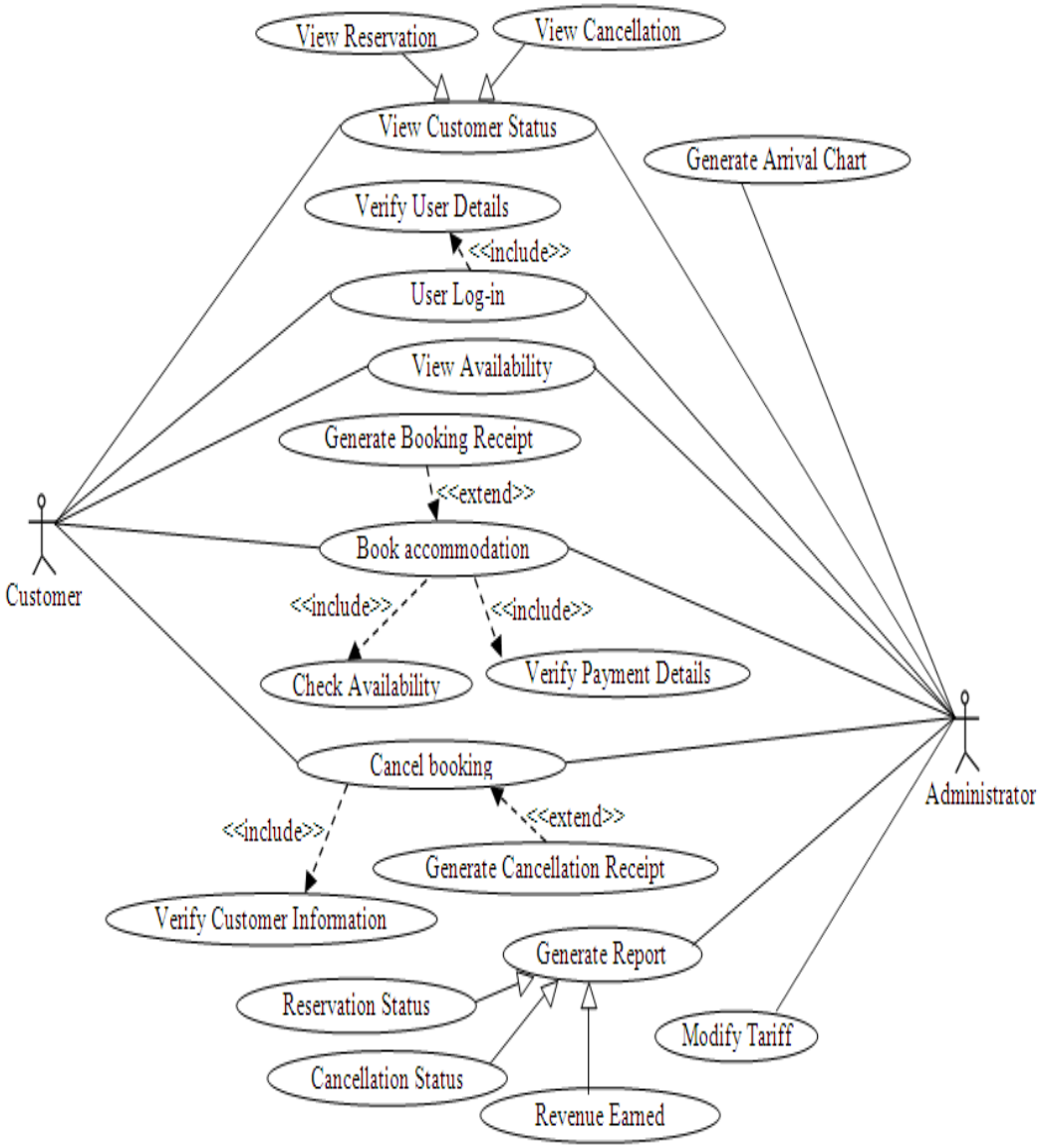

Figure 2. Use Case Diagram for OHRS [14]

The Activity diagram specifies 'how' a system would accomplish its goals. It models a business process that represents a set of coordinated tasks that achieve a business goal. Figure 3 depicts the Activity diagram for the OHRS cloud SaaS under consideration. The high-level actions are chained together to represent the business process of the OHRS.

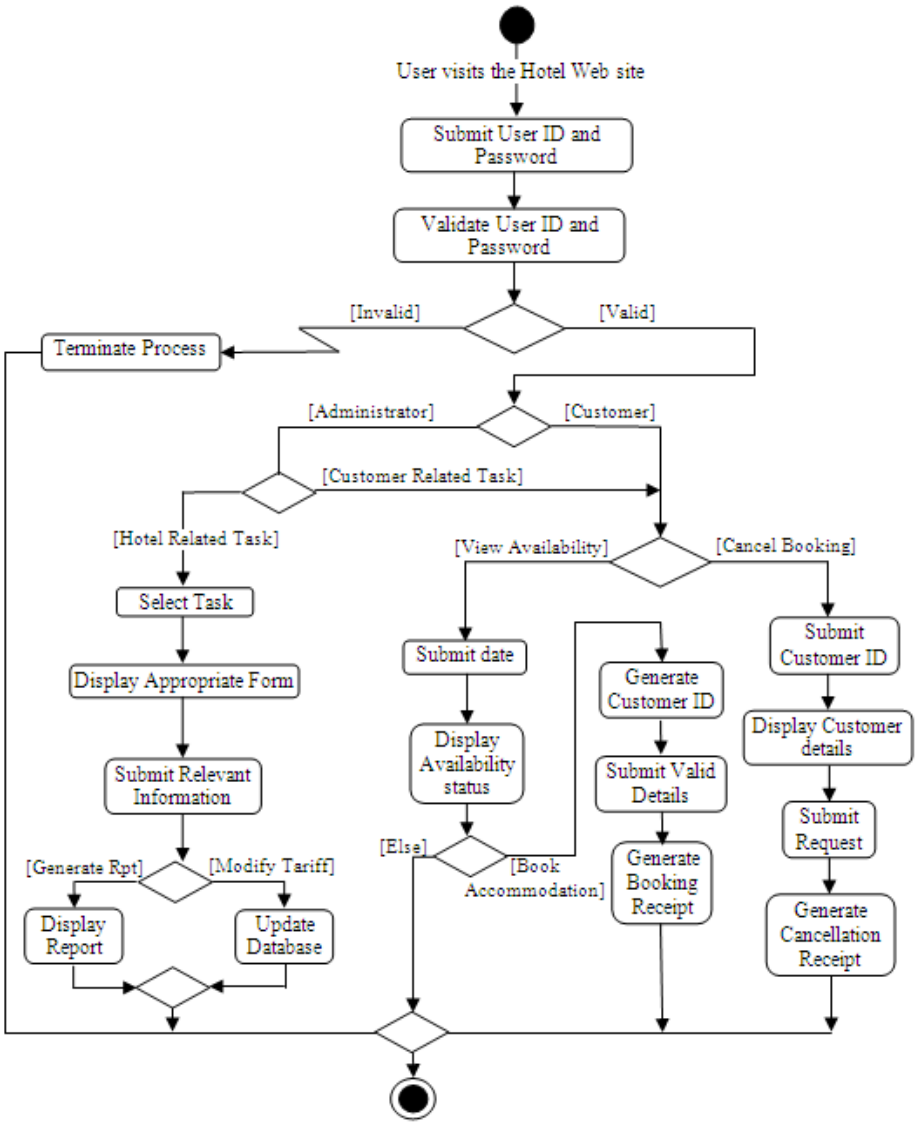

Figure 3. Activity Diagram for OHRS [14]

The various steps in the process may be listed as [14]:

1. A user (customer or administrator) logs into the hotel website with a valid user ID and password.
2. A customer may perform various tasks such as viewing the availability status, book accommodation or cancel a previously booked accommodation.
3. In order to view the availability status, the customer submits the date in response to which the system displays the availability status of the hotel for the next fifteen days.
4. The customer may then either quit the application or may proceed to book an accommodation based on its availability.
5. In response to the book accommodation action, the system generates a unique ID for the customer.

6. Next, the customer submits his details such as name, address, phone, email address, unit type, number of units, book-from date and book-to date and payment details.
7. A booking receipt bearing all the details is generated for the customer and the process terminates
8. A customer may also cancel a previous booking, in which case he submits the customer ID that was generated at the time of booking.
9. In response to step 8, the system displays the details of the customer.
10. The customer submits the cancellation request.
11. A cancellation receipt is generated for the customer and the process terminates.
12. An administrator, after successful log-in, is able to perform all the tasks that a customer does and can also generate reports or modify tariffs by submitting the required information.

Ideally, in MDA-based software development the requirements model should be simply submitted to the generators that would produce the required systems. But, in actual practice the requirements model need to be refined further into a computational model that a generator can process.

5.2. Platform Independent Model (PIM) for OHRS

The PIM of a system may be represented using a UML class diagram which exhibits classes and relationships among them. A class is a blueprint or a template for the objects that are instantiated from it. It describes two essential pieces of information – the *state* of the object represented by attributes and the *behaviour* of the object represented by operations. The classes do not exist in isolation. Instead, they work together using different types of relationships such as association, aggregation and dependency. An association between two classes means that a class contains a reference to one or more objects of the other class. Besides, an association may sometimes introduce a new class, called the association class. Aggregation and composition represent stronger version of association indicating that a class owns objects of another class. A composition is a stronger relationship than aggregation and represents the whole-part relationship between the objects.

The PIM for the OHRS cloud SaaS is depicted in Figure 4, by means class diagram. The classes, their attributes and operations, and the associations among various classes are shown in the model. The *accessor* and *mutator* methods for the attributes are not defined explicitly in the class, as the MDA mappings automatically expand the attributes into corresponding accessor and mutator operations. A multiplicity adorns each association. This PIM defines the static aspects of the OHRS application through a static view. Though this model reflects the technicalities of the system, it is non-committal to the platform that would implement and host the system.

The various classes specified in the PIM for OHRS are:

1. *Hotel* – A hotel is composed of one or more unit types. The Hotel class is therefore depicted at the *whole* end of the composition relationship between Hotel and UnitType classes. The various attributes representing the state of the hotel object include ID, name, address and phone. This class is also related to the Customer class in a one-to-many association representing that a Hotel may have zero or more customers associated with it. A one-to-many association also exists between Hotel and Administrator classes representing that the Hotel has at least one Administrator associated with it.
2. *UnitType* – The UnitType class is depicted at the *part* end of the relationship and includes the attributes such as ID, name, total_units and tariff to represent the state of its object. The class is related to Customer and Administrator classes in a many-to-many association displaying that a customer or an administrator may book one or more unit types in the hotel.

3. *Customer* – A Customer class is characterized by attributes such as ID, name, address, phone and email to represent the state of customer object. A unique ID is generated for each customer at the time of booking an accommodation. An ID relates the customer to a Hotel. The class also provides methods that enable an online customer to view the availability of rooms in the hotel, and also to book an accommodation or cancel a previously booked accommodation.
4. *Administrator* – The Administrator class has several attributes like ID, name, address, phone and email that characterize the state of administrator object. Besides, it includes the methods that facilitate viewing of availability/reservation /cancellation status of the various rooms in the hotel, book or cancel accommodation for the customer, update the hotel database and generate various reports for the management.
5. *User* – The User class with its attributes username and password is related to both Customer and Administrator classes in a one-to-one association. It also includes the methods to facilitate login and logout by the users.

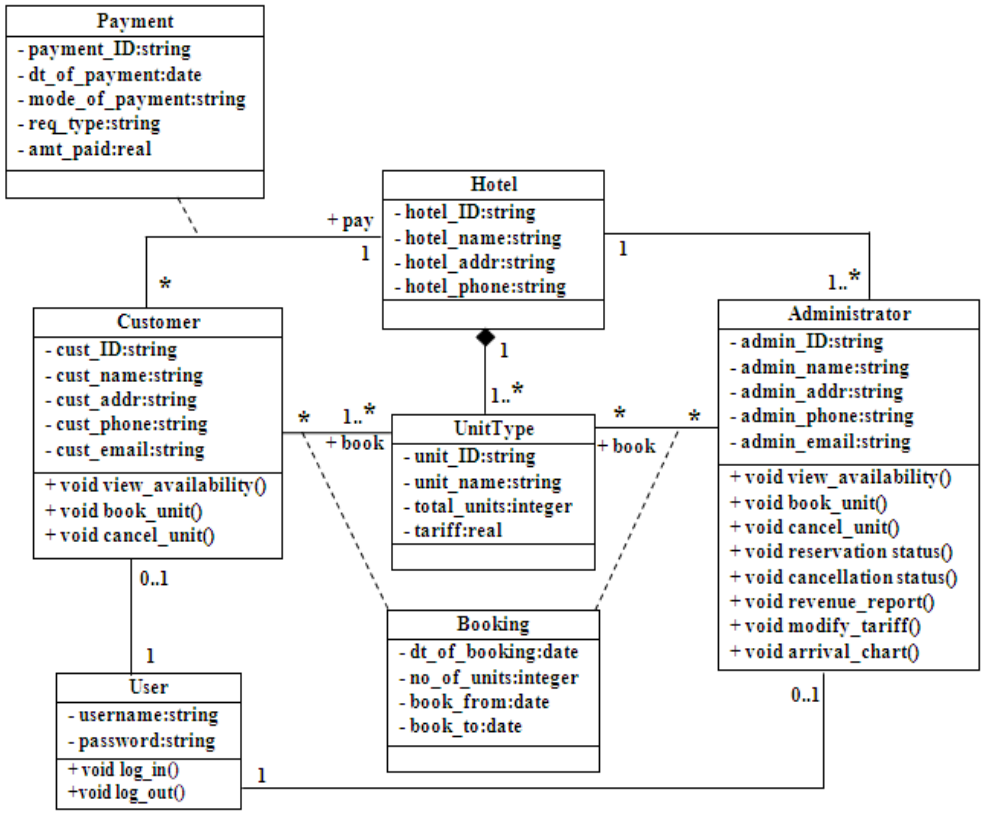

Figure 4. Platform Independent Model for OHRS [14]

Besides these classes there are two association classes – Payment and Booking – in the OHRS PIM. An association class is useful in complex cases. It is related to two classes which in turn have a relationship with each other.

6. *Payment* – This association class is related to Customer and Hotel classes, which themselves are related. The attributes of this class include payment ID, payment date, payment mode and amount paid to represent the state of the class objects.

7. *Booking* – This class represents the association between Customer-UnitType and Administrator-UnitType classes. It includes various attributes such as date of booking, number of units booked, book from date and book to date for representing the state of Booking objects.

The PIM of the OHRS cloud SaaS describes the attributes and operations in a manner that is entirely independent of any programming language or operating system in which the system would finally be implemented.

5.3. A Relational PSM for OHRS

A PSM describes the technology specific details for the target platform. One or more PSMs can be derived from a single PIM using automated transformation tools.

The PIM of OHRS is transformed into a Relational PSM based on certain transformation rules defined in section 4.4.1. The Relational PSM specifies the database and is described by a relational model depicted in an Entity-Relationship diagram in Figure 5 [14].

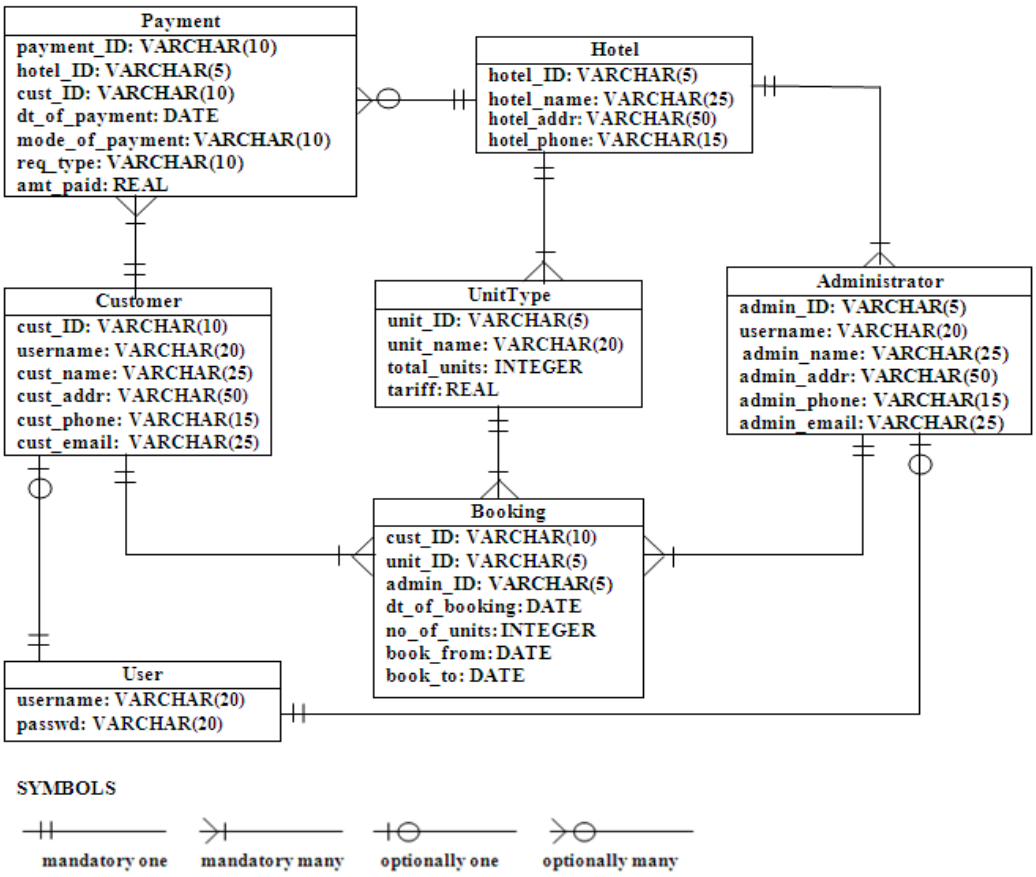

Figure 5. Relational Model for OHRS (PSM) [14]

5.3.1. Transformation Definition: PIM to Relational PSM

A transformation definition is comprised of transformation rules that describe how one or more constructs in the source language can be transformed into one or more constructs in the target

language. The transformation rules defined here take care of a consistent object-relational mapping [6]. These rules describe how the elements in PIM can be mapped to elements in Relational PSM. Based on the various elements in the PIM, the transformation rules are categorized into [14]:

Transformation Rules for Data types: A basic data type is mapped to corresponding data type in the relational model according to the following rules:

i. A UML string is mapped onto a SQL VARCHAR(n).
ii. A UML integer is mapped onto a SQL INTEGER.
iii. A UML real is mapped onto a SQL REAL.
iv. A UML date is mapped onto a SQL DATE.

A data type may be a simple data type or a derived data type.

a. Transforming Simple Data types: A simple data type is mapped directly to a column in a table.
b. Transforming Derived Data types:
   i. A UML data type that has no operations, such as struct and array, is mapped onto a number of columns, each representing a field in the derived data type.
   ii. A UML data type that is a class, is mapped to a table itself. The column holds a reference (foreign key) to a key value in the other table.

B. Transformation Rules for Classes and Attributes:

a. Each UML class is transformed into a Relation/Table with the same name.
b. Each UML attribute is transformed into a field (column) in the table with the same name.
c. When the type of attribute is not a data type but a class, the field in the table holds a foreign key to the table representing that class.

C. Transformation Rules for Associations :

a. Associations in the UML model are transformed into a foreign key relation in a database model, possibly introducing a new class.
b. An association class represents a relationship between two classes. It is transformed into a table with foreign keys in this new table referring to the two related tables.
c. The multiplicities of an association from class A to class B may be:
   - The multiplicity at A is zero-or-one,
   - The multiplicity at A is one, or
   - The multiplicity at A is more than one.

The same holds for the multiplicity at B resulting in nine different combinations of multiplicities at both the ends. A pseudocode for the rule may be expressed as:

```
if the association A to B is adorned by an association class or the multiplicity
   at both ends is more-than-one
then create a table representing the association class or the association and
   create foreign keys in this new table referring to the related tables
else if the multiplicity at one end is zero-or-one
    then create a foreign key in the table representing the class at that end,
       referencing the other end
    else /* the multiplicity of the association is one-to-one */
       create a foreign key in one of the tables, referencing the other end
    endif
endif
```

5.4. An EJB PSM for OHRS

The PIM of a cloud software application may be transformed into several PSMs targeted on different platforms. A PSM of OHRS targeted on EJB platform is specified here. The OHRS PIM is transformed into an EJB PSM based on the transformation rules specified in section 5.4.1.

A coarse-grained EJB component model is used wherein the components are large and have infrequent interactions with relatively high amount of data in each interaction. This is achieved by clustering closely related classes into one single component, the EJB data schema. With fewer components in the system, the inter-component communication would not load the network [9].

In a composite aggregation, a class that is part of a whole is clustered into the data schema that is generated from the whole. Thus, in the OHRS PIM, the class UNITTYPE is a part of the class HOTEL. Also, the association class is clustered in the data schema that is generated from the associated class that is able to navigate to the other navigated class. Therefore, in the OHRS PIM, the PAYMENT association class is clustered into the CUSTOMER class.

The granularity of the EJB components is based on the composition of classes in the domain model. As mentioned above, the class UNITTYPE is clustered with class HOTEL, and the classes PAYMENT and BOOKING are clustered with the class CUSTOMER. Figure 6 below depicts the resultant four EJB components: HOTEL, CUSTOMER, ADMINISTRATOR and USER.

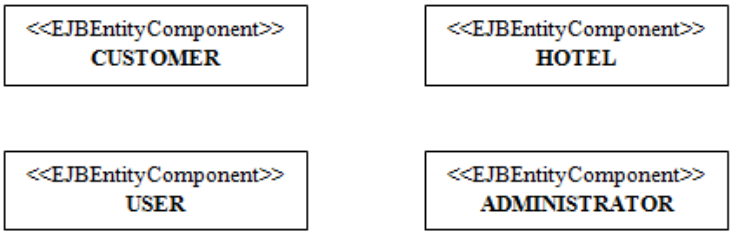

Figure 6. Top-level EJB Component Model for OHRS

Figure 7 depicts an EJB PSM for the OHRS cloud software system. The USER and ADMINISTRATOR entity components have not been depicted in the figure due to space constraints.

5.4.1. Transformation Definition: PIM to EJB PSM

The transformation rules for transforming the PIM constructs into EJB PSM constructs are listed below [9]:

1. For each PIM class, an EJB key is generated.
2. Each PIM class that is not a composite part of another PIM class is transformed into an EJB component and an EJB data schema.
3. Each PIM class is transformed into an EJB data class residing in an EJB data schema that is generated from the PIM class that is the outermost composition of the transformed PIM class.
4. Each PIM association is transformed into an EJB association within an EJB data schema that is generated from the PIM class that is the outermost composition of the transformed PIM association.

5. Each PIM association class is transformed into two EJB associations and an EJB data class. The EJB associations and the EJB data class are generated within the EJB data schema that is generated from the PIM class that is the outermost composition of the PIM class that can navigate across the transformed PIM association class.
6. Each PIM attribute of a class is transformed into an EJB attribute of the mapped EJB data class.
7. Each PIM operation is transformed into an EJB operation of the generated EJB component that is generated from the PIM class that is the outermost composition of the PIM class that owns the transformed PIM operation.

The term 'outermost composition of x' in the transformation rules refers to the class that is not a part (via a composite association) of another class and is equal to or the (direct or indirect) container of the class x.

6. CONCLUSION AND FUTURE WORK

Cloud computing is the latest buzzword in the field of Information and Communication Technology (ICT) that is based on Internet and web technologies and provides dynamically configured and massively scalable resources as services. With hardware and software technologies evolving at a tremendous pace, a major challenge faced by the IT industry in the recent years is technology obsolescence. In the context of cloud computing, this technology evolution incurs additional expenditure on part of the cloud service providers, as the applications in the cloud need to be re-engineered with newer technologies. In this regard, MDA approach is an asset which facilitates creation of good designs that easily cope with multiple-implementation technologies and extended software lifetimes. In this paper, the authors highlight the need to incorporate the MDA approach in the development of cloud SaaS in order to minimize the time, cost and efforts in application development and to enhance the Return on Investment.

The authors have lately completed the development of transformation tools based on the transformation definitions described here. The authors are presently working on an SOA-based approach for ensuring interoperability among the models of cloud software services.

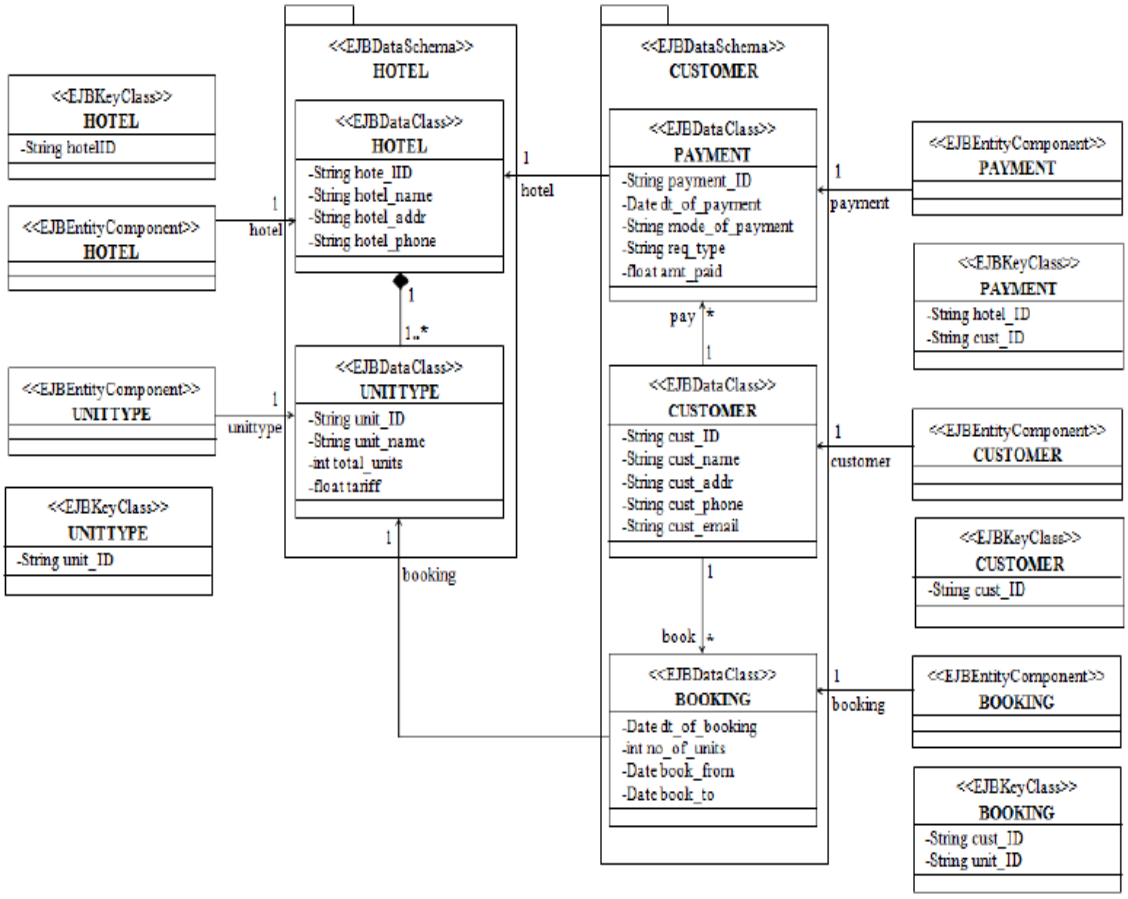

Figure 7. EJB Component Model of OHRS including the Data Schemas

REFERENCES

[1] Andy Bechtolsheim. Cloud Computing. http://netseminar.stanford.edu/seminars/Cloud.pdf (Nov 2008).

[2] Rich Maggiani: Cloud computing is changing how we communicate. In: IEEE International Professional Communication Conference, pp.1-4 (2009)

[3] Francesco Maria Aymerich, Gianni Fenu, Simone Surcis: An Approach to a Cloud Computing Network. In: First International Conference on the Applications of Digital Information and Web Technologies, pp. 113–118 (2008) ISBN: 978-1-4244-2623-2, doi: 10.1109/ICADIWT.2008.4664329

[4] Peter Mell and Tim Grance: The NIST Definition of Cloud Computing. Version 15, 10-7-09, http://thecloudtutorial.com/nistcloudcomputingdefinition.html

[5] Ian Foster, Yong Zhao, Ioan Raicu, Shiyong Lu: Cloud Computing and Grid Computing 360-Degree Compared. In: IEEE Grid Computing Environments Workshop, pp 1–10, (November 2008).

[6] Bhaskar Prasad Rimal, Eunmi Choi, Ian Lumb: A Taxonomy and Survey of Cloud Computing systems. In: Fifth International Joint Conference on INC, IMS and IDC, pp. 44-51(2009)

[7] OMG Model Driven Architecture. http://www.omg.org/mda/

[8] Joaquin Miller and Jishnu Mukerji: MDA Guide Version 1.0.1, http://www.omg.org/docs/omg/03-06-01.pdf

[9] Anneke Kleppe, Jos Warmer, Wim Bast: MDA Explained: The Model Driven Architecture: Practice and Promise. Pearson Education, Inc.(2003)

[10] Ritu Sharma and Manu Sood: Cloud SaaS and Model Driven Architecture. In: International Conference on Advanced Computing and Communication Technologies, pp. 18-22, ISBN: 978-981-08-7932-7 (2011)

[11] Sharma, R., Sood, M. 2011. Modeling Cloud Software-as-a-Service: A Perspective. In Proceedings of the International Conference on Network Communication and Computer pp 170–174. ISBN: 978-1-4244-9550-4 (2011)

[12] David Frankel, John Parodi: Using MDA to develop Web Services. IONA Technologies PLC, Second Edition, April, 2002.

[13] David S. Frankel: Model Driven Architecture: Applying MDA to Enterprise Computing. Wiley Publishing Inc, 2003.

[14] Ritu Sharma and Manu Sood: Cloud SaaS: Models and Transformation. In: Advances in Digital Image Processing and Information Technology. Communications in Computer and Information Science, 2011, Volume 205, Part 2, 305-314, DOI: 10.1007/978-3-642-24055-3_31

[15] Russ Miles and Kim Hamilton, Learning UML 2.0, O'Reilly Media Inc. August 2007.