Feature Modeling of Software as a Service Domain to Support Application Architecture Design

Karahan Öztürk
Department of Computer Engineering,
Middle East Technical University
Ankara, Turkey
e-mail: karahanozturk@gmail.com

Bedir Tekinerdogan
Department of Computer Engineering
Bilkent University
Ankara, Turkey
e-mail: bedir@cs.bilkent.edu.tr

***Abstract*—Cloud computing is an emerging computing paradigm that has gained broad interest in the industry. SaaS architectures vary widely according to the application category and number of tenants. To define a proper SaaS architecture it is important to have a proper understanding of the domain. Based on our extensive domain analysis approaches, we provide a feature model for SaaS that depicts the design space and represents the common and variant parts of SaaS architectures. The feature model enhances the understanding of SaaS systems, and supports the architect in designing the SaaS application architectures.**

Keywords- modeling, service, architecture, design, SaaS

I. INTRODUCTION

Cloud computing is an emerging computing paradigm that has gained broad interest [6][19]. Unlike traditional enterprise applications that rely on the infrastructure and services provided and controlled within an enterprise, cloud computing is based on services that are hosted on providers over the Internet. The services that are hosted by cloud computing approach can be broadly divided into three categories: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service and Software-as-a-Service (SaaS). In this paper we will focus on the Software as a Service context [18]. SaaS is a web-based, on-demand distribution model where the software is hosted and updated on a central site and does not reside on client computers [1][3]. With SaaS, software applications are rented from a provider as opposed to purchased for enterprise installation and deployment. Similar to the general benefits of cloud computing the SaaS approach yields benefits such as reduced cost, faster-time-to-market and enhanced scalability.

An appropriate SaaS architecture design will play a fundamental role in supporting the cloud computing goals [13][4]. Based on the literature we can derive the basic components required for SaaS. However, while designing particular applications one may derive various different application design alternatives [1] for the same SaaS architecture specification. Each design alternative may meet different functional and nonfunctional requirements. It is important to know the possible design so that a viable realization can be selected.

To enhance the understanding of SaaS systems and support the architect in designing SaaS architectures we propose defining a feature model for SaaS architectures. A feature model is the result of a domain analysis process whereby the common and variant properties of a domain or product are elicited and modeled [15]. In addition, the feature model identifies the constraints on the legal combinations of features and as such, a feature model defines the feasible models in the domain. The feature model has been derived after an extensive literature study to SaaS architectures. This included basically a systematic literature study on cloud computing in general and software as a service architectures in particular. It should be noted that we could not put all the references in this paper due to space limitations. Based on a commonality and variability analysis of the selected papers the common and variant features of SaaS were derived.

The remainder of the paper is organized as follows. Section II presents SaaS architecture for which a feature model will be defined. Section III presents the family feature model for SaaS. Section IV presents an example illustrating the derivation of application architecture based on application feature model. Finally section V concludes the paper.

II. SOFTWARE AS A SERVICE ARCHITECTURE

SaaS has been widely discussed in the literature and various definitions have been provided. In general when describing SaaS, no specific application architecture is prescribed but rather the general components and structure is defined. Based on the literature we have defined the reference architecture for SaaS as given in Figure 1 [3][13][18][6]. Besides of the theoretical papers we have also looked at documentation of reference architectures as defined by SaaS vendors such as Intel [18], Sun [19] and Oracle [10].

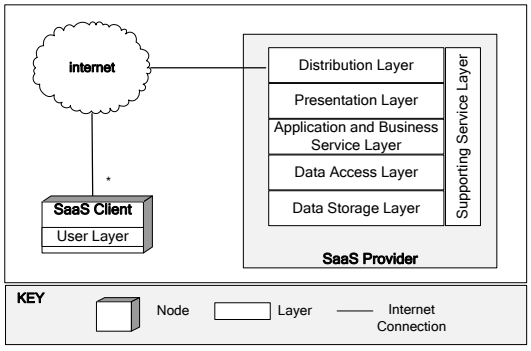

Figure 1. SaaS Reference Architecture

 ISBN: 978-1-61208-165-6

In principle, SaaS has a multi-tier architecture with multiple thin clients. In Figure 1 the multiplicity of the client nodes is shown through the asterisk symbol (*). In SaaS systems the thin clients rent and access the software functionality from providers on the internet. As such the cloud client includes only one layer User Layer which usually includes a web browser and/or the functionality to access the web services of the providers. This includes, for example, data integration and presentation. The SaaS providers usually include the layers of Distribution Layer, Presentation Layer, Business Service Layer, Application Service Layer, Data Access Layer, Data Storage Layer and Supporting Service Layer.

Distribution Layer defines the functionality for load balancing and routing. *Presentation Layer* represents the formatted data to the users and adapts the user interactions. The *Application and Business Service Layer* represents services such as identity management, application integration services, and communication services. *Data Access Layer* represents the functionality for accessing the database through a database management system. *Data Storage Layer* includes the databases. Finally, the *Supporting Service Layer* includes functionality that supports the horizontal layers and may include functionality such as monitoring, billing, additional security services, and fault management. Each of these layers can be further decomposed into sub-layers.

Although Figure 1 describes the common layers for SaaS reference architecture, it deliberately does not commit on specific *application architecture*. For example, the number of clients, the allocation of the layers to different nodes, and the allocation of the data storage to nodes is not defined in the reference architecture. Yet, while designing SaaS for a particular context we need to commit on several issues and make explicit design decisions that define the application architecture. Naturally, every application context has its own requirements and likewise these requirements will shape the SaaS application architecture in different ways. That is, based on the SaaS reference architecture we might derive multiple application architectures.

III. FEATURE MODEL OF SAAS

To support the architect in designing an appropriate SaaS application architecture a proper understanding of the SaaS domain is necessary. In this section we define the SaaS feature model that represents the overall SaaS domain. Figure 2 shows the conceptual model representing the relation between feature model and SaaS architecture.

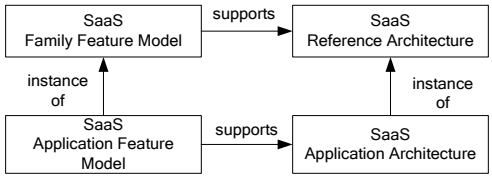

Figure 2. Conceptual model representing relation between feature model and SaaS architecture

We distinguish between *family feature model* and *application feature model*. The family feature model represents the features of the overall SaaS domain, whereas the application feature model represents the features for a particular SaaS project. The application feature model is derived from the family feature model. The features in the feature model typically refer to the architectural elements in the SaaS architecture. As discussed in the previous section we also distinguish between SaaS reference architecture and SaaS application architecture. For designing the SaaS application architecture first the required features need to be selected from the family feature model resulting in the application feature model. The application feature model will be used to support the design of the SaaS application architecture. In the following we will elaborate on the family feature model.

A. Top-Level Feature Model

The top level feature diagram of SaaS that we have derived is shown in Figure 3. The key part represents the different types of features including *optional*, *mandatory*, *alternative*, and *or features* [15]. Note that the features in Figure 3 denote the layers in the SaaS reference architecture as defined in Figure 1. All the layers except the Support Layer have been denoted as mandatory features. The Support Layer is defined as optional since it might not always be provided in all SaaS applications. Each of these layers (features) can be further decomposed into sub-layers.

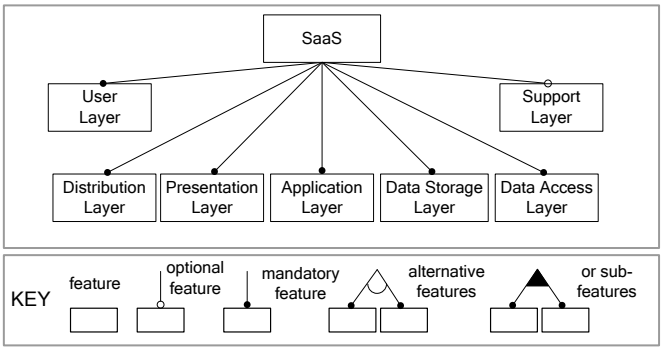

Figure 3. Top-Level Feature Model

B. User Layer

User layer is the displaying layer that renders the output to the end user and interacts with the user to gather input. This layer is the only part that the user can see. In principle the user layer might include a *Web Browser* or Rich Internet Application (RIA), or both of these (or features). RIA is especially used on mobile platforms.

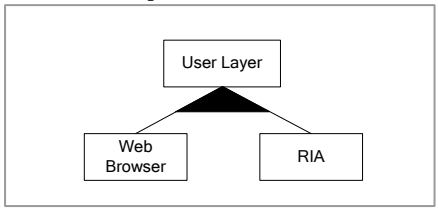

Figure 4. Feature Diagram for User Layer

C. Distribution Layer

Figure 5 shows the features for the distribution layer feature. This layer is the intermediate layer between the

 ISBN: 978-1-61208-165-6

internet and the SaaS application. The main concerns of the layer are scalability, availability and security. The mandatory features of this layer are load balancers and firewalls [11].

A firewall inspects the traffic and allows/denies packets. In addition to this, firewalls provide more features like intrusion detecting, virtual private network (VPN) and even virus checking. The distribution layer can have a single firewall or a firewall farm. A firewall farm is a group of connected firewalls that can control and balance the network traffic.

Load balancers divide the amount of workload across two or more computers to optimize resource utilization and increase response time. Load balancers are also capable of detecting the failure of servers and firewalls and repartitioning the traffic. Load balancers have the mandatory features of *Type* and *Strategy*, and an optional feature *Load Balancer.Firewall*. There are two types of load balancers, *hardware based* and *software based*. Load balancing strategies decide how to distribute requests to target devices. *Passive* load balancing strategies use already defined strategies regardless the run time conditions of the environment. Some of the most used passive strategies are *Round Robin*, *Failover*, *Random* and *Weighted Random*. *Dynamic* load balancing strategies are aware of information of the targets and likewise route the requests based on traffic patterns. Some of the most used passive strategies are *Fastest Response Time*, *Least Busy*, *Transfer Throughput*, *IP Sticky* and *Cookie Sticky*.

The optional *Load Balancer.Firewall* can be used as firewall by providing both packet filtering and stateful inspection. Using load balancer as a firewall can be an effective solution for security according to network traffic and cost requirements. This feature excludes the "*Distribution Layer.Firewall*" feature.

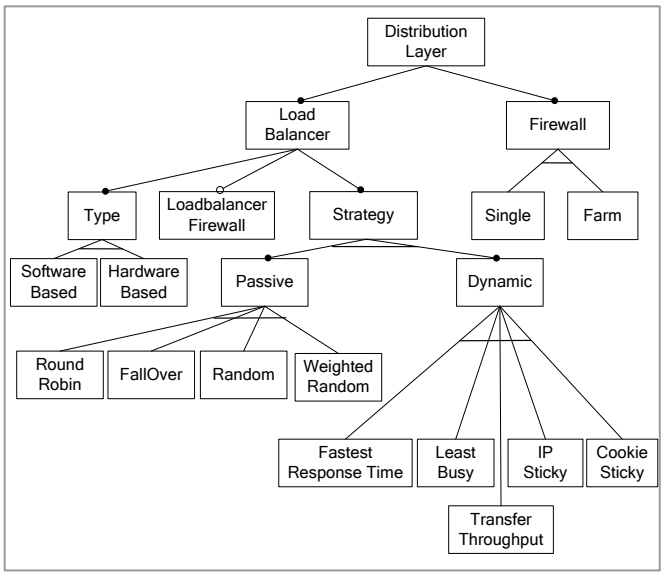

Figure 5. Feature Diagram for Distribution Layer

D. Presentation Layer

Figure 6 presents the presentation layer feature. The presentation layer consists of components that serve to present data to the end user. This layer provides processes that adapt the display and interaction for the client access. It communicates with application layer and is used to present data to the user.

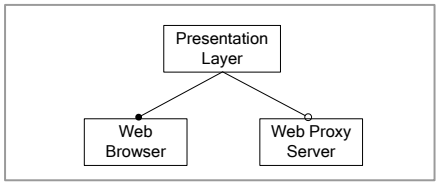

Figure 6. Feature Diagram for Presentation Layer

The presentation layer feature includes two subfeatures, the mandatory *Web Server* and optional *Web Proxy Server* features. A web server handles HTTP requests from clients. The response to this request is usually an HTML page over HTTP. Web servers deal with static content and delegate the dynamic content requests to other applications or redirect the requests. *Web Proxy Server* can be used to increase the performance of the web servers and presentation layer, caching web contents and reducing load is performed by web proxy servers. Web proxy servers can also be used for reformatting the presentation for special purposes as well for mobile platforms.

E. Application Layer

Figure 7 shows the feature diagram for Application Layer, which is the core layer of the SaaS architecture. Business logic and main functionalities, Identity Management, orchestration, service management, metadata management, communication, and integration are provided by this layer.

Especially in the enterprise area, SaaS platforms are usually built on SOA technologies and web services. *Application Server*, *Integration*, *Metadata Management*, *Identity Management* and *Communication* are mandatory features for the application layer. In case of using SOA, some other features – *ESB*, *Orchestration*, *Business Rules Engine*, are used in this layer. In the following subsections we describe these features in more detail.

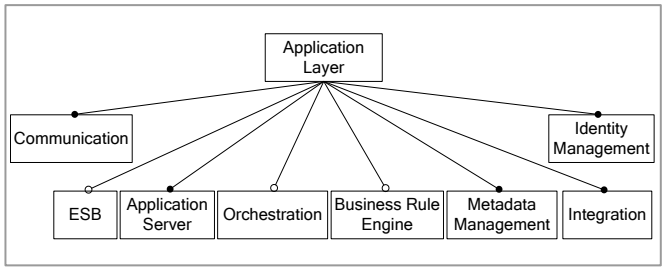

Figure 7. Feature Diagram for Application Layer

- *Application Server*

An application server is a server program that handles all application operations between users and an organization's backend business applications or databases. The application server's mission is to take care of the business logic in a multi-tier architecture. The business logic includes usually the functions that the software performs on the data. Application servers are assigned for specific tasks, defined by business needs. Its basic job is to retrieve, handle, process and present

 ISBN: 978-1-61208-165-6

data to the user interface, and process any input data whether queries or updates, including any validation and verification and security checks that need to be performed.

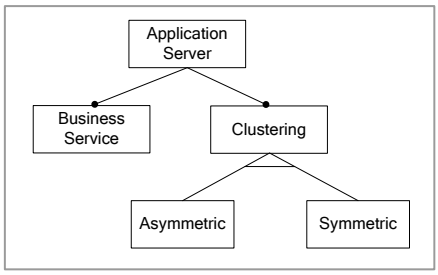

Figure 8. Feature Diagram for Application Server

SaaS applications have to have continuous uptime. Users around the world can access the application anytime. Application failure means customer and monetary loss. The application should be prevented from single point of failure. In addition to availability issues, there are performance and scalability capabilities to overcome for SaaS applications. By combining more than one computer and make it as a unified virtual resource can solve these problems. This technique is called server clustering. There are two techniques for server clustering: *asymmetric* and *symmetric*. In asymmetric clusters, a standby server exists to take control in case of another server gets of failure. In symmetric clusters, every server in the cluster do actual job. The first technique provides more available and fault tolerant system but the latter is more cost-effective.

- *ESB*

When we are talking about SaaS applications and service oriented architecture, the requirement is providing an infrastructure for services to communicate, interact, and transform messages. Enterprise Service Bus (ESB) is a platform for integrating services and provides enterprise messaging system. Using an ESB system does not mean implementing a service oriented architecture but they are highly related and ESB facilitates SOA.

- *Orchestration*

Orchestration is a critical mission in SOA environment. A lot of tasks should be organized to perform a process. Orchestration provides the management, coordination and arrangement of the services. BPEL is, for example, an orchestration language that defines business processes. Some simple tasks may be performed by ESB but more complex business processes could be defined by BPEL. To interpret and execute BPEL a BPEL engine is needed.

- *Metadata Management*

SaaS has a single instance, multi-tenant architecture. Sharing the same instance to many customers brings the problem of customization. In SaaS architecture, customization is done using metadata. Metadata is not only about customization (e.g. UI preferences), it is also intended to provide configuration of business logic to meet customers need. Updating, storing and fetching metadata is handled through Metadata services. This feature requires *Metadata Repository* feature.

- *Business Rule Engine*

As mentioned before, SaaS applications can be customized and configured by metadata. Workflow may differ for each customer. Business Rules Engine is responsible of metadata execution. It consists of its own rule language, loads the rules and then performs the operations.

- *Integration*

The feature diagram for Integration is shown in Figure 9. In the context of SaaS, all the control, upgrade, and maintenance of user applications and data are handled by SaaS provides. An important challenge in SaaS is the data integration. SaaS applications usually need to use client data which resides at the client's node. On the other hand, each client may use more than one SaaS application or on-premise application using the same data. The data may be shared among several applications and each application may use different part of it or in different formats. Manipulating the data will usually have an impact on the other applications. Data accuracy and consistency should be provided among those applications. Re-entering or duplicating the data for any application is not a feasible manner to provide data.

There are three different approaches for providing consistent data integration including: *common integration*, *specific integration* and *certified partner integration*. In the common integration approach services are provided for all clients. This feature requires "*Integration.Services.Web Services*" feature. In the specific integration, services are customized for each customer. This feature requires "*Integration.Services.Integration Services*" feature. Finally, in the *Certified Partner* approach the SaaS vendor delegates the integration to another vendor which is a specialist for SaaS integration. The SaaS vendor still needs to provide web services, but it leaves the control to other entities and focuses itself on the application. This feature also requires "*Integration.Services.Web Services*" feature.

The Integration feature describes either *Integration Service* or *Web Service*: In *Integration Service* approach, the SaaS vendor provides custom integration services for customers. Although this is the easiest way for customers, it is hard to manage adding integration service for different needs for vendors and increasing number of customers causes scalability problems. In the *Web Service* approach, the SaaS vendor provides a standard approach for customers as web services. The customers themselves take responsibility for SaaS integration. Compared to the Integration Service approach, customers have to do much more and need extensive experience. On the other hand this is a more scalable solution for vendors.

 ISBN: 978-1-61208-165-6

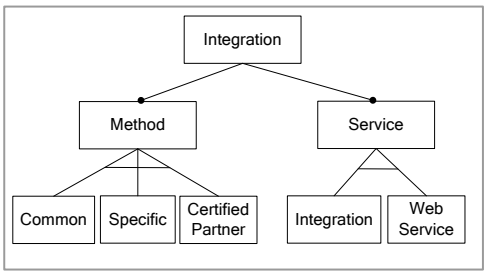

Figure 9. Feature Diagram for Integration

- *Identity Management*

Figure 10 represents the feature model for Identity Management, which deals with identifying individuals in a system and controlling access to the resources in the system by placing restrictions on the established identities of the individuals [7]. The *Directory Management* is responsible for managing the identities.

Identify Management includes two mandatory features *Identity Model* and *Directory Management*. Identity Model can be *Single Sign-On*, *Isolated* or *Federated*. *Isolated Identity Management*: The most common and simplest identity management model is the isolated one. Hereby, each service provider associates an identity for each customer. Despite its simplicity, this model is less manageable in case of the growth of number of users who should remember their login and passwords to their accounts for each service. Single Sign-On is a centralized identity management model, which allows users to access different systems using a single user ID and password.

Single Sign-On identity management model [5] can be *PKI-Based*, *SAML-Based*, *Token-Based*, *Credential Synchronization*, or Secure *Credential Caching*. SAML stands for Security Assertion Markup Language and defines the XML based security standard to enable portable identities and the assertion of these identities. The *Token-Based* approach can be either based on *Kerberos* or *Cookie*. The *Secure Credential Caching* can be on the *Server Side* or *Client Side*.

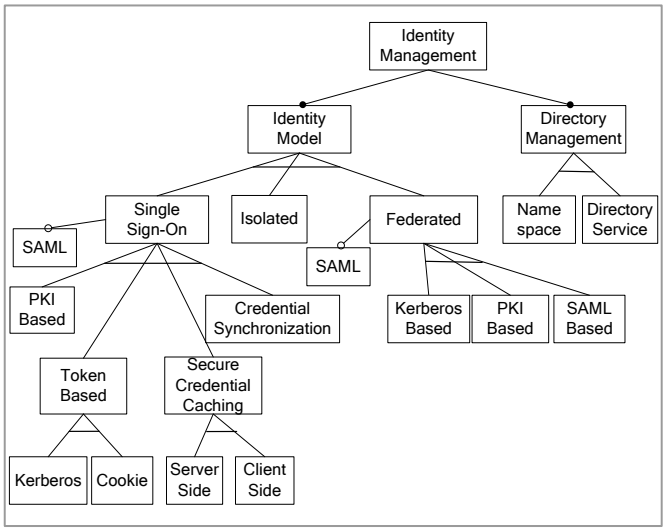

Figure 10. Feature Diagram for Identity Management

The *Federated Identity Model* is very close to Single Sign-On, but defined identity management across different organizations [6]. There are three most used approaches, *Kerberos-based Federation*, *PKI-based Federation* or *SAML-based Federation*. *Directory Management* feature includes two mandatory features, *Namespace* and *Directory Service. Namespace* maps the names of network resources to their corresponding network addresses. *Directory Service* represents the provided services for storing, organizing and providing access to the information in a directory (e.g. *LDAP).*

- *Communication*

Figure 11 shows the feature model for the *Communication* feature. SaaS vendor needs to provide a communication infrastructure both for inbound and outbound communication. Notification, acknowledging customers, sending feedbacks, demanding approvals are useful for satisfying users. The most common approach for communication is e-mailing. To transfer mails between computers a *Mail Transfer Agent (MTA)* can be used which requires *Simple Mail Transfer Protocol (SMTP)* protocol. Besides of mailing other protocols such as *Short Message Peer-to-Peer Protocol (SMPP)* and *Simple Network Paging Protocol (SNPP)* can be used.

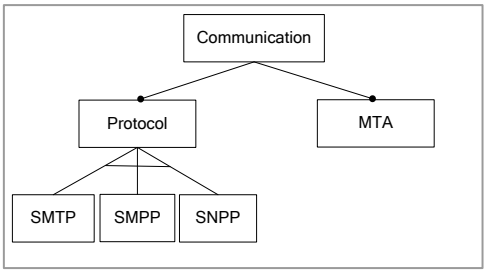

Figure 11. Feature Diagram for Communication

F. Data Access Layer

Figure 12 shows the feature diagram for *Data Access Layer*. This layer provides the database management system (DBMS) consisting of software which manages data (database manager or database engine), structured artifact (database) and metadata (schema, tables, constraints etc.).

One of the important, if not the most important, SaaS feature is multi-tenancy [2][12]. Multi tenancy is a design concept where a single instance of software is served to multiple consumers (tenants). This approach is cost saving, scalable, easy to administrate, because the vendor has to handle, update or upgrade and run only single instance. Multi-tenancy is not only about data, this design can be applied in all layers but the most important part of the multi tenancy is multi tenant data architecture. Based on the latter different kind of multi-tenancy can be identified. Multi-tenancy with *Separate Databases* means that each tenant has its own data set which is logically isolated from other tenants. The simplest way to data isolation is storing tenant data in separate database servers. This approach is best for scalability, high performance and security but requires high cost for maintenance and availability. In the *Shared Database, Separate Schemas* approach, a single database server is used for all tenants. This approach is more cost effective but the main disadvantage is

 ISBN: 978-1-61208-165-6

restore is difficult to achieve. Finally, the *Shared Database, Shared Schema* approach involves using one database and one schema for each tenants' data. The tables have additional columns, tenant identifier column, to distinguish the tenants. This approach has the lowest hardware and backup costs.

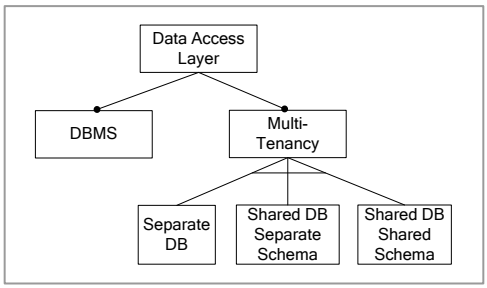

Figure 12. Feature Diagram for Data Access Layer

G. Data Storage Layer

Figure 13 shows the feature diagram for *Data Storage Layer*. The layer includes the feature for Metadata storage, Application Database and Directory Service. Metadata files can be stored either in a database or in a file based repository. Application Database includes the sub-features of Storage Area Network (*SAN*), *Clustering* and *Caching* [2]. SAN is a dedicated storage network that is used to make storage devices accessible to servers so that the devices appear as locally attached to the operating system. SAN is based on fiber channel and moves the data between heterogeneous servers.

Clustering is interconnecting a group of computers to work together acting like a single database to create a fault-tolerant, high-performance, scalable solution that's a low-cost alternative to high-end servers. By caching, disk access and computation are reduced while the response time is decreased.

Directory Service stores data in a directory to let the directory service to lookup for identity management. This data is read more often than it is written and can be redundant if it helps performance. Directory schemas are defined as object classes, attributes, name bindings and namespaces.

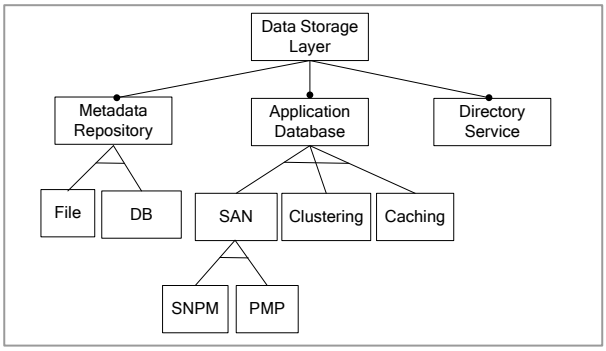

Figure 13. Feature Diagram for Data Storage Layer

H. Supporting Service Layer

Supporting Service Layer is a cross-cutting layer that provides services for all layers. The feature model is shown in Figure 14. As known, SaaS applications have quality attributes such as scalability, performance, availability and security. To keep the applications running efficiently and healthy, the SaaS system needs to have monitoring system to measure metrics. The monitoring infrastructure can detect failures, bottlenecks, and threats and alert the administrators or trigger automatic operations. Furthermore, SaaS systems may be built on service oriented architecture and may need metering process for service level agreements and billing. A few examples for the metrics are CPU usage, CPU load, network traffic, memory usage, disk usage, attack rate, number of failures, mean time to respond etc.

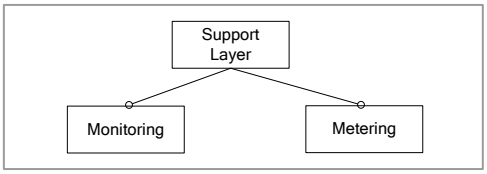

Figure 14. Feature Diagram for Support Layer

IV. EXAMPLE

Figure 15 shows an alternative application architecture design that is derived from the reference architecture shown in Figure 1. To derive this architecture based on the family feature model as discussed in the previous sections, the application feature model is defined. Typically in the application feature model multi-tenancy is selected using a single database management system with a shared database and shared schemas for the tenants.

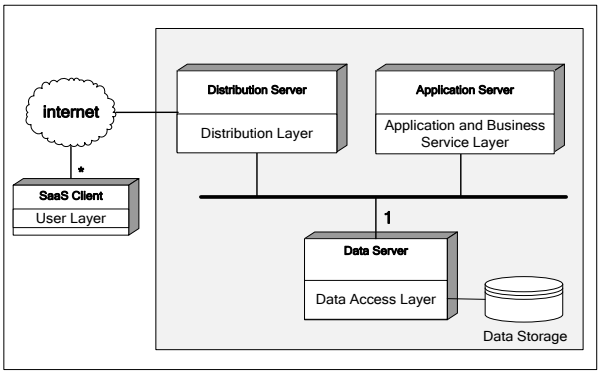

Figure 15. SaaS Application Architecture derived based on corresponding application feature model

V. RELATED WORK

Despite its relatively young history, different surveys have already been provided in the literature on cloud computing and many papers have been published on SaaS. An example survey paper is provided by Goyal and Dadizadeh [8]. However, to the best of our knowledge no systematic domain analysis approach has been carried out to derive a feature model for SaaS.

La and Kim [14] propose a systematic process for developing SaaS systems highlighting the importance of reuse. The authors first define the criteria for designing the process model and then provide the meta-model and commonality and variability model. The metamodel defines the key elements of SaaS. The variability model is primarily represented as a table. The work focuses more on the general approach. The metamodel could be complementary to the reference

 ISBN: 978-1-61208-165-6

architecture in this paper and as presented by SaaS providers. Although the goal seems similar, our approach appears to be more specific and targeting the definition of a proper modeling of the domain using feature modeling.

Godse and Mulik [9] define an approach for selecting SaaS products from multiple vendors. Since the selection of the feasible SaaS product involves the analysis involves analysis of various decision parameters the problem is stated as a multi-criteria decision-making (MCDM) problem. The authors adopt the Analytic Hierarchy Process (AHP) technique for prioritizing the product features and for scoring of the products. The criteria that are considered in the AHP decision process are *Functionality*, *Architecture*, *Usability*, *Vendor Reputation*, and *Cost*. Our work is also focused on selecting the right SaaS product but it considers the design of the SaaS architecture based on feature modeling. The selection process defines the selection of features and not products. However, in our approach we did not outline the motivation for selecting particular features. For this we might add additional criteria to guide the architect also in selecting the features. We consider this as part of our future work.

Nitu [16] indicates that despite the fact that SaaS application is usually developed with highly standardized software functionalities to serve as many clients as possible, there is still a continuous need of different clients to configure SaaS for their unique business needs. Because of this observation, SaaS vendors need take a well designed strategy to enable self serve configuration and customization by their customers without changing the SaaS application source code for any individual customer. The author explores the configuration and customization issues and challenges to SaaS vendors, and distinguishes between configuration and customization. Further a competency model and a methodology framework is proposed to help SaaS vendors to plan and evaluate their capabilities and strategies for service configuration and customization. The work of Nitu considers the configuration of the system after the system architecture has been developed. We consider our work complementary to this work. The approach that we have presented focuses on early customization of the architecture to meet the individual client requirements. The approach as presented by Nitu could be used in collaboration with our approach, i.e. by first customizing the architecture based on the potential clients and then providing configurability and customization support for the very unique business needs.

VI. CONCLUSION

Cloud computing and SaaS is a broad domain that is not easy to understand for novice designers. In this paper we have applied domain analysis techniques to derive a family feature model that represents both the common and variant features of SaaS architecture. Based on the family feature model a particular application feature model can be derived and the SaaS application architecture can be designed accordingly. As such, the family feature model helps both to enhance the understandability of SaaS and the generation of particular applications.

The feature model that we have derived is based on our selection of papers. We do not claim that this is the only correct or eventual feature model. Enhancing the domain analysis study might refine the feature model that we have presented. Yet, the work should also be considered from an architecture design perspective. An important lesson from this paper is that feature modeling helps to support the architectural design of SaaS systems. In our future work we will develop the required tool support to represent the family feature model, define the link with architecture design decisions and generate application architecture.

VII. REFERENCES

[1] S. A. Brandt, E. L. Miller, D. D. E. Long, L. Xue. Efficient Metadata Management in Large Distributed Storage Systems, 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies(MSST'03), pp. 290–298, 2003.

[2] F. Chong and G. Carraro. Building Distributed applications: Multi-Tenant Data Architecture. MSDN architecture center, 2006.

[3] F. Chong and G. Carraro. Architecture Strategies for Catching the Long Tail, Microsoft, MSDN architecture center, 2006.

[4] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford. Documenting Software Architectures: Views and Beyond. Second Edition. Addison-Wesley, 2010.

[5] J. de Clercq, Single Sign-On Architectures, Proceedings of the International Conference on Infrastructure Security, p.40-58, October 01-03, 2002.

[6] Cloud Computing. Wikipedia - [Online]. http://en.wikipedia.org/wiki/Cloud_computing

[7] FIDIS, "Structured Overview on Prototypes and Concepts of Identity Management Systems", Future of Identity in the Information Society (No. 507512)

[8] A. Goyal, S. Dadizadeh. A Survey on Cloud Computing, University of British Columbia, Technical Report, 2009.

[9] M. Godse, S. Mulik. An Approach for Selecting Software-as-a-Service (SaaS) Product, in Proc.of. 2009 IEEE International Conference on Cloud Computing, 2009.

[10] S. Joshi. Architecture for SaaS applications - using the Oracle SaaS Platform, Oracle White Paper, 2009.

[11] C. Kopparapu, "Load Balancing Servers, Firewalls, and Caches", Wiley, 2002.

[12] T. Kwok, T. Nguyen. A Software as a Service with Multi-tenancy Support for an Electronic Contract Management Application. In IEEE International Conference on Services Computing, 2008.

[13] P.A. Laplante, Jia Zhang, Jeffrey Voas, "What's in a Name - Distinguishing between SaaS and SOA", IT Professional, Volume 10, Issue 3 (May 2008), Pages: 46-50, Year of Publication: 2008,

[14] H. Jung La and Soo Dong Kim, A Systematic Process for Developing High Quality SaaS Cloud Services, in Proc. Proc. of the 1st International Conference on Cloud Computing, Springer LNCS, Volume 5931/2009, 278-289, 2009.

[15] K. Lee , K. Chul Kang , J. Lee, Concepts and Guidelines of Feature Modeling for Product Line Software Engineering, Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools, p.62-77, April 15-19, 2002

[16] H. Liao. Design of SaaS-Based Software Architecture, International Conference on New Trends in Information and Service Science, 2009.

[17] Nitu. ISEC '09: Proceeding of the 2nd annual conference on India software engineering conference, , pp. 19-26, February 2009.

[18] C. Spence, J. Devoys, S.Chahal. Architecting Software as a Service for the Enterprise IT@Intel White Paper, 2009.

[19] Sun Cloud Computing Primer, http://www.scribd.com/doc/54858960/Cloud-Computing-Primer, accessed 2011.

 ISBN: 978-1-61208-165-6