# MODELING OF SOFTWARE AS A SERVICE ARCHITECTURES AND INVESTIGATION ON THEIR DESIGN ALTERNATIVES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

KARAHAN ÖZTÜRK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JUNE 2010

Approval of the thesis:

MODELING OF SOFTWARE AS A SERVICE ARCHITECTURES AND INVESTIGATION ON THEIR DESIGN ALTERNATIVES

submitted by **KARAHAN ÖZTÜRK** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı _____
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Ali H. Doğru _____
Supervisor, **Computer Engineering Dept., METU**

Asst. Prof. Dr. Bedir Tekinerdoğan _____
Co-advisor, **Computer Engineering Dept., Bilkent University**

**Examining Committee Members:**

Prof. Dr. Adnan Yazıcı _____
Computer Engineering Dept., METU

Assoc. Prof. Dr. Ali H. Doğru _____
Computer Engineering Dept., METU

Asst. Prof. Dr. Pınar Şenkul _____
Computer Engineering Dept., METU

Dr. Cevat Şener _____
Computer Engineering Dept., METU

Snr. Eng. Abdullah ÖZTÜRK _____
MilSOFT Software Corp.

**Date:** 25.06.2010

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

**Name, Last Name** : Karahan ÖZTÜRK

**Signature** :

# ABSTRACT

MODELING OF SOFTWARE AS A SERVICE ARCHITECTURES AND INVESTIGATION ON THEIR DESIGN ALTERNATIVES

Öztürk, Karahan

M.S., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Ali H. Doğru

Co-advisor: Asst. Prof. Dr. Bedir Tekinerdoğan

June 2010, 110 pages

In general, a common reference architecture can be derived for Software as a Service (SaaS) architecture. However, while designing particular applications one may derive various different application design alternatives from the same reference SaaS architecture specification. To meet the required functional and nonfunctional requirements of different enterprise applications it is important to model the possible design so that a feasible alternative can be defined. In this thesis, we propose a systematic approach and the corresponding tool support for guiding the design of the SaaS application architecture. The approach defines a SaaS reference architecture, a family feature model and a set of reference design rules. Based on the business requirements an application feature model is defined using the family feature model. Selected features are related to design decisions and a SaaS application architecture design is derived. By defining multiple application architectures based on different application feature models we can even compare multiple alternatives and based on this select the most feasible alternative.

**Keywords:** Cloud Computing, Software as a Service (SaaS), Feature Model, Domain Analysis, Reference Architecture, Model Driven Architecture, Deployment Diagram.

# ÖZ

SERVİS OLARAK YAZILIM MİMARİLERİN MODELLENMESİ VE TASARIM ALTERNATİFLERİNİN İNCELENMESİ

Öztürk, Karahan
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Ali H. Doğru
Ortak Tez Yöneticisi: Yar. Doç. Dr. Bedir Tekinerdoğan

Haziran 2010, 110 sayfa

Genellikle, Servis olarak Yazılım (SoY – SaaS) için referans mimari oluşturulsa da, belirli türdeki uygulamalar için aynı referans mimariden bir çok tasarım alternatifi türetilebilir. Farklı türdeki kurumsal uygulamaların işlevsel ve işlevsel olmayan gereksinimlerini karşılamak için olası tasarımları modellemek, uygulanabilir alternatifleri ortaya çıkarmak açısından önemlidir. Bu tezde, SaaS uygulama mimarisini tasarlamak için sistematik bir yaklaşım öneriyor ve gerekli araç desteğini sağlıyoruz. Bu yaklaşım, SaaS referans mimarisini, ana özellik modelini ve bir referans tasarım kararı kümesini tanımlar. İş gereksinimleri taban alınarak, ana özellik modelinden, uygulama özellik modeli türetilir. Seçilen özellikler tasarım kararlarıyla ilişkilidir ve böylece SaaS uygulama mimari tasarımı ortaya çıkarılmış olunur. Birden çok uygulama mimarisi geliştirerek, alternatifler arasından en uygun olanını seçilebilir ve bu mimariler arasında nitelik bazında muhakeme yapılabilir.

**Anahtar Kelimeler:** Bulut İşletim, Servis olarak Yazılım (SoY – SaaS), Özellik Modeli, Alan Çözümlemesi, Referans Mimari, Model Odaklı Mimari, Dağıtım Çizeneği.

***To Azime...***

# ACKNOWLEDGMENTS

I would like to present my deepest thanks to my thesis supervisor Assoc. Prof. Dr. Ali H. Doğru and co-advisor Assoc. Prof. Dr. Bedir Tekinerdoğan for their valuable guidance, motivation and support throughout this thesis study.

I am very grateful to my family for all their patience and tolerance.

My special thanks go to Aysu Betin Can, Mustafa Azak and Abdullah Çetin Çavdar for their help and support to complete this work and to all my friends who gave me support whenever I needed.

# TABLE OF CONTENTS

# LIST OF TABLES

TABLES

# LIST OF FIGURES

FIGURES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| **SaaS** | Software-as-a-Service |
| **PaaS** | Platform-as-a-Service |
| **IaaS** | Infrastructure-as-a-Service |
| **ADL** | Architecture Description Language |
| **MDA** | Model Driven Architecture |
| **DSL** | Domain Specific Language |
| **GMF** | Graphical Modeling Framework |
| **FODA** | Feature Oriented Domain Analysis |
| **SOA** | Service Oriented Architecture |
| **SSO** | Single Sign On |
| **ESB** | Enterprise Service Bus |
| **DBMS** | Database Management System |
| **XML** | Extensible Markup Language |

# CHAPTER 1

# INTRODUCTION

Cloud computing is an emerging computing paradigm that has gained world wide interest [3][13]. Dissimilar to 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. In cloud computing, services are fully managed by the provider, and consumers can buy the required amount of services on demand, use applications without installation and access their personal files at any computer over internet access. The central hosting of both the application and data of consumers allows for more flexible, effective, and efficient computing. In last years the interest and use of cloud computing have been accelerated with the significant developments in virtualization and distributed computing, as well as improved access to high-speed Internet and the need for economical optimization of resources.

The services that are hosted by cloud computing approach can be generally 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 study we will focus on the Software as a Service context [69][70][71]. SaaS is a web-based, on-demand distribution model where the software is owned, hosted and updated at a central site and does not reside on client computers. SaaS seems to be the most developed category of cloud computing, because it evolved from the application-service-provider model of software hosting. With SaaS, software

applications are rented from a vendor as opposed to purchased for enterprise installation and deployment. Like the general benefits of cloud computing the SaaS approach yields benefits such as reduced cost, faster-time-to-market and enhanced scalability.

Obviously, an appropriate SaaS architecture design will play a fundamental role in supporting the cloud computing goals. 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 [63] 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.

In this study, based on a domain analysis process [65] we define a reference architecture for SaaS that represents the common components and their interactions of various SaaS platforms. Based on the reference architecture, we propose an approach for (1) modeling the design space of SaaS application design alternatives (2) and guiding the selection of these design alternatives based on the particular requirements. The approach consists of five steps: First, a reference feature model [65] is defined for SaaS architecture that defines the possible features of SaaS as defined by the literature. Second, based on the feature model, required design decisions are defined and mapped to SaaS feature model. Third, an application model is selected from the feature model for a desired SaaS application architecture and matching design decisions are extracted. Fourth, design decisions are converted to an internal-use architecture description language (ADL) [71]. Fifth, a diagram editor is provided and deployment diagram is generated from the ADL instance, so that, by the visual representation, design of the architecture will be quite easier and understandable.

The remainder of the thesis is organized as follows.

**Chapter 2 – Background** gives an overview of cloud computing and Software as a Service in literature. It introduces the main paradigms of cloud computing, and identifies the characteristics, usage, and benefits in detail. This chapter also describes the SaaS architecture.

**Chapter 3 – Feature Model of SaaS** presents a new approach for modeling and analyzing SaaS architecture. Common and variable features of SaaS architecture and the relationship between the features are described.

**Chapter 4 – Deriving SaaS Application Architectures** describes the approach for deriving various SaaS application architectures, based on the feature model of SaaS.

**Chapter 5 – Tool Support** provides information about the tools that we have developed to perform the processes mentioned in chapter 3 and chapter 4.

**Chapter 6 – Conclusions and Feature Work** discusses the concluding remarks and explains the future work.

# CHAPTER 2

# BACKGROUND

This chapter aims to present an overview of cloud computing. It also includes the general concepts of SaaS architectures with key elements and benefits.

## 2.1 Cloud Computing

In many ways, cloud computing is simply a metaphor for the Internet, the increasing movement of compute and data resources onto the Web. But there's a difference: cloud computing represents a new break point for the value of network computing. It delivers higher efficiency, large scalability, and faster, easier software development. It's about new programming models, new IT infrastructure, and the enabling of new business models and markets [59].

Cloud computing gets its name as an analogy for the Internet. Typically, the Internet is represented in network diagrams as a cloud in diagrams. The cloud icon represents "all that other stuff" that makes the network work. Details are abstracted from the users. It also takes the expertise and control over. It's likely this notion that is most applicable to the cloud computing concept [58].

A cloud typically consists of a complete application stack-servers, storage, networking and applications with all the relevant dependencies that users can

subscribe to. One would merely pay for the resources they consume, with all the parts and pieces being managed by the service provider behind the scene.

Alternatively, a cloud could involve just an individual component or service such as a storage cloud (e.g., Amazon S3) or a payment gateway (e.g., Amazon Simple Pay), and the customer is responsible for tying in all the requisite components from different providers to form a complete cloud. Cloud is defined as being open, flexible, efficient, pre-designed, standardized and highly automated.

Cloud computing is a model for enabling favorable, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model defines the paths ahead in computer science and information technologies worlds. Based on decades of research it utilizes all recent achievements in virtualization, distributed computing, utility computing, and networking [58] [59] [61].

### 2.1.1 Characteristics

- **On-demand Self Service:** A consumer can one-sidely provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service's provider [61].

- **Broad Network Access:** Capabilities are reachable over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, and PDAs) [61].

- **Resource Pooling:** The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model by assigning the physical and

virtual resources according to consumer demand. There is a notice of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or data center). Examples of resources such as storage, processing, memory, network bandwidth, and virtual machines [61].

- **Rapid Elasticity:** Capabilities can be rapidly and flexibly provided, in some cases automatically, to quickly scale out, and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be limitless and can be purchased in any quantity at any time [61].

- **Measured Service:** Cloud systems automatically control and optimize resource use by supplying a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported for both the provider and consumer of the utilized service by providing transparency [61].

### 2.1.2 Benefits

Cloud computing brings a new level of efficiency, effectiveness and economy to delivering IT resources on demand — and in the process it opens up some new business models and market opportunities.

While most of people think of current cloud computing offerings as purely “pay by the use” compute platforms, they’re really a union of two major interdependent IT trends:

- **IT Efficiency:** Reduce and minimize costs where companies are converting their IT costs from capital expenses to operating expenses through technologies such as virtualization. Cloud computing begins as a model to improve infrastructure resource deployment and utilization, but fully achieving this infrastructure eventually leads to a new application development model [59].

- **Business Agility:** Improve and maximize return using IT as a competitive weapon through rapid time to market, integrated application stacks, instant machine image deployment, and large scale parallel programming. Cloud computing is wrapped as a critical way to revolutionize time to service. But without doubt these services must be built on equally innovative rapid-deployment-infrastructure models [59].

To be sure, these progressions and trends have existed in the IT industry for years. Nevertheless, the recent emergence of massive network bandwidth and virtualization technologies has enabled this transformation to a new services-oriented infrastructure.

Cloud computing enables IT organizations to increase hardware utilization rates severely, and to scale up to massive capacities in an instant — without constantly having to invest in new infrastructure, train new personnel, or license new software. It also creates new occasions to build a better breed of network services, in less time, for less money.

- **IT Efficiency**

Cloud computing is completely about efficiency. It provides an approach to deploy and access everything from single systems to huge amounts of IT

resources — on demand, in real time, at an affordable cost. It makes very high-performance compute and high-capacity storage available to anyone with a credit card. And since the best cloud strategies build on concepts and tools that developers already know, clouds additionally have the potential to redefine the relationship between information technology and the developers and business units that depend on it [59].

**Reduce capital expenditures**: Cloud computing makes it possible for companies to make the conversion of IT costs from capital expense to operating expense through technologies such as virtualization. The cloud promises to reduce and minimize the cost of acquiring, delivering, and maintaining computing power, a benefit of particular importance in times of fiscal uncertainty. By enabling agencies to purchase merely the computing services needed, instead of investing in complex and expensive IT infrastructures, agencies can drive down the costs of developing, testing, and maintaining new and existing systems [59].

**Cut the cost of running a datacenter:** Cloud computing improves infrastructure utilization percentages and streamlines resource management. For example, clouds allow for self-service provisioning resources through APIs, bringing a higher level of automation to the datacenter and reducing management costs [61].

**Resource Maximization:** Cloud computing makes it easy to burden on IT resources already stretched thin, particularly important for agencies facing shortages of qualified IT professionals [61].

**Scalability and Capacity:** Cloud computing provides scaling on demand, which, when combined with utility pricing (pay as you go), removes the need to overprovision to meet demand. With cloud computing, companies can scale up to large capacities in an instant. The cloud is an always-on computing resource that

enables users to tailor consumption of resources to their specific needs. Infinitely scalable, cloud computing allows IT infrastructures to be expanded effectively, efficiently and expediently without the necessity of making major capital investments. Capacity can be added as resources are required and completed in a very short period of time. Thus, agencies can prevent the latency, expense, and risk of purchasing hardware and software that takes up data center space - and can reduce the traditional time required to scale up an application in support of the mission. Cloud computing allows agencies to move in the other direction as well without difficulty, removing capacity, and thus expenses, as needed [59][60].

- **Faster, More Flexible Programming**

Cloud computing isn't solely about hardware — it's also a programming revolution. Agile, easy-to-access, lightweight Web protocols — coupled with horizontally scaled architecture — can accelerate development cycles rapidly and time to market with new applications and services. New business capabilities are now just a script away [59].

**Access:** The cloud promises global access to high-powered computing and storage resources for anyone with a network access device. By providing such functions, cloud computing helps to support the continuity of operations demands [61].

**Collaboration:** The cloud provides an environment where users can develop software-based services that enhances collaboration and encourage greater information sharing, and not only within the agency, but also among other government and private entities [61].

**Customization:** Cloud computing offers an environment for creating and tailoring applications to address a diversity of tasks and challenges. Cloud brings

agility, which means that specific processes can be easily changed to meet shifting needs, because those processes are typically changeable by making a configuration change, and not by driving redevelopment from the back-end systems [58][59][61].

### 2.1.3 Deployment Models

#### 2.1.3.1 Private Cloud

The cloud infrastructure is operated only for an organization. It may be controlled by the organization or a third party and may exist on premise or off premise [61].

#### 2.1.3.2 Community Cloud

The cloud infrastructure is shared through several organizations and supports a specific community that has shared concerns and constraints (e.g., mission, security requirements, policy, and compliance considerations). It may be controlled by the organizations or a third party and may exist on premise or off premise [61].

#### 2.1.3.3 Public Cloud

The cloud infrastructure is made reachable to the general public or a large industry group and is owned by an organization selling cloud services [61].

#### 2.1.3.4 Hybrid Cloud

The cloud infrastructure is a composition of two or more clouds mentioned above (private, community, or public) that remain unique entities but are bound together

by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds) [61].

### 2.1.4 Layers of Cloud Computing

While the first revolution of the Internet saw the multi tiers, three-tier (or n-tier) model emerge as a general architecture, the use of virtualization in clouds has created a new set of layers: applications, services, and infrastructure. These layers don't just encapsulate on-demand resources; in addition to that they also define a new application development model. And within each layer of abstraction there are innumerable business opportunities for defining services that can be offered on a pay-per-use basis [59].

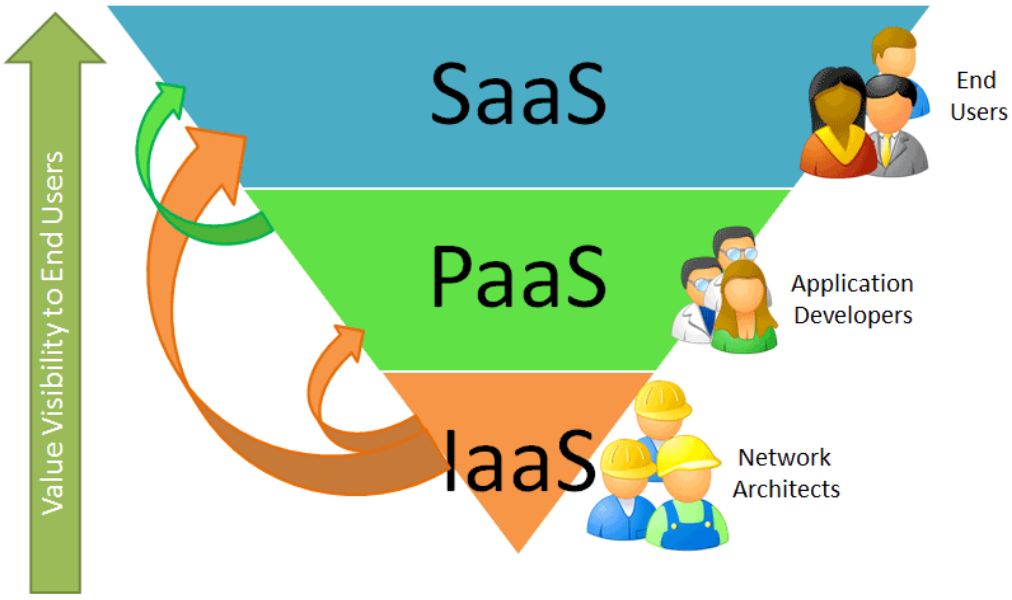


**Figure 1 – Layers of cloud computing**

#### 2.1.4.1 Software as a Service (SaaS)

SaaS is at the highest, most mature layer and features a complete application offered as a service, on demand, via multi tenancy — meaning a single instance of the software runs on the provider's infrastructure and serves multiple client organizations. The most broadly known example of SaaS is Salesforce.com, but there are now many others, including the Google Apps offering of basic business services such as e-mail. Of course, Salesforce.com's multi-tenant application has preceded the definition of cloud computing just by a few years. On the other hand, like many other players in cloud computing, Salesforce.com now operates at more than one cloud layer with its release of another service, Force.com, a companion application development environment, or platform as a service [59].

#### 2.1.4.2 Platform as a Service (PaaS)

PaaS, this service lies on the middle layer, is the encapsulation of a development environment abstraction and the packaging of a payload of services. The archetypal payload is a Xen image (part of Amazon Web Services) containing a basic Web stack (for example, a Linux distribution, a Web server, and a programming environment such as Pearl or Ruby).

PaaS offerings can leverage every phase of software development and testing, or they can be specialized around a specific area, such as content management.

Some enterprise examples include Google App Engine, which serves applications on Google's infrastructure. PaaS services such as these can leverage a great deal of flexibility but may be constrained by the capabilities that are available through the provider [59].

#### 2.1.4.3 Infrastructure as a Service (IaaS)

IaaS, lies on the bottom, is at the lowest layer and is a means of delivering basic storage and compute functionalities as standardized services over the network. Servers, storage systems, switches, routers, caches and other systems are pooled (through virtualization technology, for example) to handle specific types of workloads — from batch processing to server/storage improving during peak loads.

The most known enterprise example is Amazon Web Services, whose EC2 and S3 services provide computing and storage services. Another example is Joyent whose main product is a set of virtualized servers which provide a highly scalable on-demand infrastructure for running Web sites and rich Web applications written in Ruby on Rails, PHP, Python, and Java [59].

#### 2.1.4.4 Cloud Computing Maturity Model

The establishment of a cloud computing maturity model (CCMM) provides a framework for successful implementation, including five key components [61]:

- Consolidation
- Virtualization
- Automation
- Utility
- Cloud

#### 2.1.4.5 Consolidation

An agency's migration towards to cloud computing starts with the consolidation of server, storage, and network resources, which works to reduce redundancy, decrease wasted space, and increase equipment usage rate, all through the measured planning of both architecture and process. Consolidation is achieved primarily through virtualization but can also be approached by the use of better computing hardware or even high performance computing. By boosting the speed of critical processes and enabling greater flexibility and efficiency, the consolidation of data centers and desktops allows agencies to do more with fewer resources – a significant concern in today's economic environment. Additionally, the shift to a unified fabric provides both physical and virtual access to the storage area network (SAN), creating greater efficiency and cost savings by allowing more storage to be consolidated in the SAN. Network and application modernization is also an important initial step in enabling the transition to a cloud computing environment and paradigm. A reasonable alternative to replacing infrastructure components or rewriting critical applications, modernization promotes communication between older systems and newer solutions, all while preserving the value in existing IT systems. Liberated from the relations of a mainframe environment, critical applications modernized through a service-oriented architecture provide agencies with the increased ability to leverage newer technologies. As for security concerns encompassing cloud computing, modernization actually works to enhance the security of sensitive information stored on critical applications. When established appropriately, the cloud platform provides security of all data in motion, traveling between the cloud and the desktop, and all data at rest in cloud storage.

#### 2.1.4.6 Virtualization

Virtualization forms a complete foundation for all cloud architectures. It enables the abstraction and aggregation of all data center resources, thus creating a unified resource that can be shared by all application loads. Hardware such as servers, storage devices, network environment and other components are treated as a pool of resources rather than a discrete system, so that allowing the allocation of resources on demand. By decoupling the physical IT infrastructure from the applications and services being hosted by service provider, virtualization allows greater efficiency and flexibility, without any effect on system administration productivity or tools and processes. By separating the workload from the underlying OS and hardware, virtualization allows extreme portability and flexibility. When extended to every system component, desktop, network, storage, and servers – virtualization enables the mobility of applications and data, not only across servers and storage arrays, but also across data centers and networks. Furthermore, through consolidation – one of the critical applications of virtualization – agencies can attain control of their distributed resources by creating shared pools of standardized resources that enable centralized management, speeding up service provisioning and reducing unplanned down time. Eventually, the result is increased use of assets and simplified lifecycle management through the mobility of applications and data. Although many agencies turn to virtualization to improve resource usage and decrease both capital and operating costs, the final goal in cloud computing is the use of the abstraction between applications and infrastructure to manage IT as a Service (IaaS) in a true cloud environment [60].

#### 2.1.4.7 Automation

In this level of maturity, automation optimizes an agency's virtualized IT resources. By this procedure, the infrastructure is automated, and critical IT processes become more dynamic - and greater control is achieved by trusted policies. With automation, data centers can systematically remove manual task requirements for run-time operations. Surrounded by the various forms of automation in practice today, provisioning automation is perhaps the best known and most often implemented one. Rather than managing underlying infrastructure, agencies in search of cloud computing need to move toward managing service levels based on what is appropriate for the application users, whether it's minimum tolerable application latency or the availability level of an application – whatever is considered critical factors. Regarding to this, automation becomes a crucial element. With centralized IT and self-service for end users, automation helps agencies to disconnect themselves from the burden of repetitive management procedures, all while enabling end users to rapidly access what they require. Finally, automation can help agencies to reduce their operating expenses by [60]:

- Reallocating computing resources in case of demanded
- Establishing run-time reactions to capacity demands
- Automating problem-ticket responses (or eliminating problem tickets for most automated response scenarios)
- Integrating system management and measurement

#### 2.1.4.8 Utility

In addition to automation, both self service and metering - feedback about the cost of the resources those are allocated - are essential requirements in creating a cloud

service. With improvement capabilities for end users and agencies, self service and metering help not only better IT management but the further extension of the user experience. In the cloud, there is no ambassador between the user of a resource and the processes for obtaining and allocating resources for critical mission requirements and initiatives. Since the user triggers the serviced requests, IT becomes an on demand service and the costs of operation drop significantly, because costs are occurred only when the service is used and less money is spent attending to the needs of the infrastructure. Constitutive to IT administration is the question of how to maintain service delivery in a fully virtualized, multi-tenancy environment while at the same time providing the highest levels of security – most importantly for information and services that might leave the data center. A private cloud utility model solves this problem, by enabling agencies to retain the data within their network security while scaling and expanding as user demands change, pooling IT resources in a single operating system or management platform. Consequently, anywhere from tens to thousands of applications and services can be supported – and new architectures that target large-scale computing activities easily installed [60].

#### 2.1.4.9 Cloud

Through cloud internetworking federation, disparate cloud systems can be linked in such a way as to accommodate both the particular nature of cloud computing and the running of IT workloads. This federation allows the sharing of a range of IT resources and capabilities – including capacity, monitoring, and management – and the movement of application loads between clouds. Moreover, since federation can occur across data center and agency boundaries, it enables such processes as unified metering and billing and one-stop self-service provisioning. With cloud computing, communication increases significantly, as data sharing between previously separate systems is fully enabled – and collaboration within

and between government agencies grows exponentially. Ultimately, rather than each agency operating in isolation, constricted by the boundaries of its own data center, not only can services be shared among groups, but also costs can be shared and lessened [60].

### 2.1.5 Security Concerns

Here the most important issue is security and the way that the provider has to assure the user of providing it. Also one of the most important aspects of cloud in which academia is more interested is high performance and adding securing will always reduce performance. Thus there is a need to find a way of implementing security with the least effect on performance.
Here are seven of the specific security issues that customers should raise with vendors before selecting a cloud vendor [60].

- **Privileged user access:** Sensitive data processed outside the enterprise brings with it an inherent level of risk, because outsourced services bypass the "physical, logical and personnel controls" IT shops exert over in-house programs. Get as much information as you can about the people who manage your data. "Ask providers to supply specific information on the hiring and oversight of privileged administrators, and the controls over their access" [60].

- **Regulatory compliance:** Customers are ultimately responsible for the security and integrity of their own data, even when it is held by a service provider. Traditional service providers are subjected to external audits and security certifications. Cloud computing providers who refuse to undergo this scrutiny are "signaling that customers can only use them for the most trivial functions" [60].

- **Data location:** When you use the cloud, you probably won't know exactly where your data is hosted. In fact, you might not even know what country it will be stored in. Ask providers if they will commit to storing and processing data in specific jurisdictions, and whether they will make a contractual commitment to obey local privacy requirements on behalf of their customers [60].

- **Data segregation:** Data in the cloud is typically in a shared environment alongside data from other customers. Encryption is effective but isn't a cure-all. "Find out what is done to segregate data at rest". The cloud provider should provide evidence that encryption schemes were designed and tested by experienced specialists. "Encryption accidents can make data totally unusable, and even normal encryption can complicate availability" [60].

- **Recovery:** Even if you don't know where your data is, a cloud provider should tell you what will happen to your data and service in case of a disaster. "Any offering that does not replicate the data and application infrastructure across multiple sites is vulnerable to a total failure". Ask your provider if it has "the ability to do a complete restoration, and how long it will take."

- **Investigative support:** Investigating inappropriate or illegal activity may be impossible in cloud computing. "Cloud services are especially difficult to investigate, because logging and data for multiple customers may be co-located and may also be spread across an ever-changing set of hosts and data centers. If you cannot get a contractual commitment to support specific forms of investigation, along with evidence that the vendor has already successfully supported such activities, then your only safe assumption is that investigation and discovery requests will be impossible." [60].

- **Long-term viability:** Ideally, your cloud computing provider will never go broke or get acquired and swallowed up by a larger company. But you must be sure your data will remain available even after such an event. "Ask potential providers how you would get your data back and if it would be in a format that you could import into a replacement application." [60].

## 2.2 SaaS

In brief, SaaS can be defined as below:
"Software deployed as a hosted service and accessed over the Internet." [11]

SaaS is an on-demand delivery model. The clients get the service by the web-based access. The application runs on the provider's side the clients do not need to install the application.

Because of the software is hosted by the service provider, the user do not care of the maintenance, upgrade, infrastructure, hardware of the service. It is an important advantage to legacy software applications. This brings the users economic and time saving.

So the SaaS is not just a delivery model, it is also a business model. In this model, the customers have access via thin clients (e.g. web browsers). The software vendor deals with delivering the service, support, upgrade, maintenance.

SaaS has a pay-as-you-go utilization model. The billing is made on subscriptions or licenses. It is generally paid monthly. Multiple users are charged on per use.
SaaS also brings economic to vendors. They will have continuous revenue stream.
SaaS has a multi-tenant architecture, which means there is running a single instance of application and many users consuming this application as a service. Customization is an important part of the application due to having single instance software and multiple users.

After the definitions, it can be thought that all SaaS services are complex business applications. But according to definition, even an email application is a concrete example of SaaS [4] [10] [11] [12].

### 2.2.1 Characteristics and Key Elements

Characteristics of SaaS software are as below [14]:

- network-based access to software
- activities managed from central locations (vendor's site) rather than at each customer's site, enabling customers to access applications remotely via Internet
- application delivery typically closer to a one-to-many model (single instance, multi-tenant architecture) than to a one-to-one model (as the ASP's manner), including architecture, pricing, partnering, and management characteristics
- centralized feature updating, which remedies the need for end-users to download patches and upgrades
- frequent integration into a larger network of communicating software systems.

Most of the SaaS platforms have common elements as below [15]:

- multi-tenancy
- ordering and provisioning
- user authentication and authorization
- service catalog and pricing
- service monitoring
- integration
- usage metering
- billing
- invoicing and payments
- support

### 2.2.2 Benefits for Users

SaaS is changing the whole IT world and this may be a dramatic end for the packaged application business. Around the world, customers are now wants to avoid the main problems they have traditionally come across when acquiring application software, including [16]:

- Having to pay for large software license fees
- Assuming the entire risk that the application will deliver the expected benefits to the business
- Having to buy expensive computer hardware and infrastructure to run the application on client-side
- Making continuous payments for hardware and software maintenance contracts
- Having to employ expensive IT experts to look after the portfolio of on-premises applications
- Losing time for negotiating the software license agreement and/or dealing with very difficult vendors

Using software-as-a-service (SaaS) via the Web, by paying a periodic subscription, avoids all of the above problems.

In addition to obviating the problems listed, the SaaS delivery model brings the benefits as below [12] [17]:

- Financial — SaaS is subscribed to and not purchased; instead of offering to the end users less costly and more predictable monthly fees, with no capital expense and initial high pay.
- Ease of use — SaaS applications can be easily accessed via Internet where it is available through a web browser, therefore taking advantage of attributes like usability and innovation of the web.

- Uptime — Server applications are hosted in a highly-available data center, minimizing the effects of power turnovers and Internet turnovers from incidents that affect local offices.
- Faster Time to Market — Cloud and SaaS services can be brought to market rapidly using automation service modules.
- Installation Ease and Low Maintenance — upgrades and patches to latest versions take short time and reduce resource requirements.
- Access Anywhere — customers can connect to their applications any where over a Web connection.
- Smaller Storage Requirements — the user does not need store software or data stored on his computer so he doesn't need large data storage accommodations. There is also the benefit of not needing to constantly backup data - storage is the responsibility of the SaaS provider.
- Fewer Personnel — SaaS reduces the need for IT experts to handle maintenance, monitoring and software updates. The SaaS vendor will provide specialists to handle these tasks.

### 2.2.3 Benefits for Vendors

The SaaS model does not rely on sales of permanent licenses and instead uses the subscription model. This reduces some of the problems with permanent licensing. There is no competition between the present and future versions of software since the vendor does not sell permanent licenses. Therefore the publisher does not need to hold back new features for the next version and subscribers will have a higher willingness to pay if they expect to receive further enrichments to software features. This shifts a vendor’s encouragement of invest in software development by allowing him to release new features as soon as they are finished and make

them available to all subscribers. So, a key advantage offered by SaaS is that individual features can be released as soon as they are completed whereas the permanent licensing model requires them to be withheld until a new version of the software is completed [7].

Another important advantage for the vendor lies in the ongoing stream of income which will amount to much more than what can be expected in the traditional software licensing setup. Through SaaS, moreover, vendors can reduce piracy and unlicensed use of software and reduce losses associated with such activities [17].

There are many other good reasons for moving to a software service model. Vendors may want to:

- Move a company with a traditional business model to a more Internet based model which offers better online services and information to customers.
- Limit the time and money costs of configuring software for customers, suppliers and internal users.
- Attract new investors, customers and partners.

### 2.2.4 SaaS vs. ASP

SaaS as a concept is often associated and compared with the application service providers (ASPs) of the 1990s, which provided "shrink-wrap" applications to business users over the Internet. These early attempts at Internet-delivered software model had more in common with traditional on-premise applications than with modern SaaS applications in some ways, such as licensing and architecture. Since these applications were originally built as single-tenant

applications, namely one-to-one model, their ability to share data and processes with other applications was limited, and they tended to offer few economic benefits over their locally installed counterparts [18].
Taking a traditional application package and providing it as a hosted service (legacy ASP-Hosting model) is not SaaS. Even adding virtualization does not make it SaaS.

In ASP model, if a hosting company has 100 tenants, then it has 100 separate instances of a software package, there are 100 software instances to install, maintain and (worst of all) customize. This model is simply not scalable and fails at unexpectedly low customer numbers [19].

Despite SaaS appears similar to ASP, it differs in the provision granularity and supply network size. SaaS coordinates the composition and arrangement of fine-grained, customized services as opposed to the ASP approach's larger-grained, more standardized applications. Also, though the ASP supply network typically pairs one customer with one supplier, the SaaS approach deploys a far larger supplier network that aggregates services into increasingly larger units until it delivers the top-level functionality requested [2].

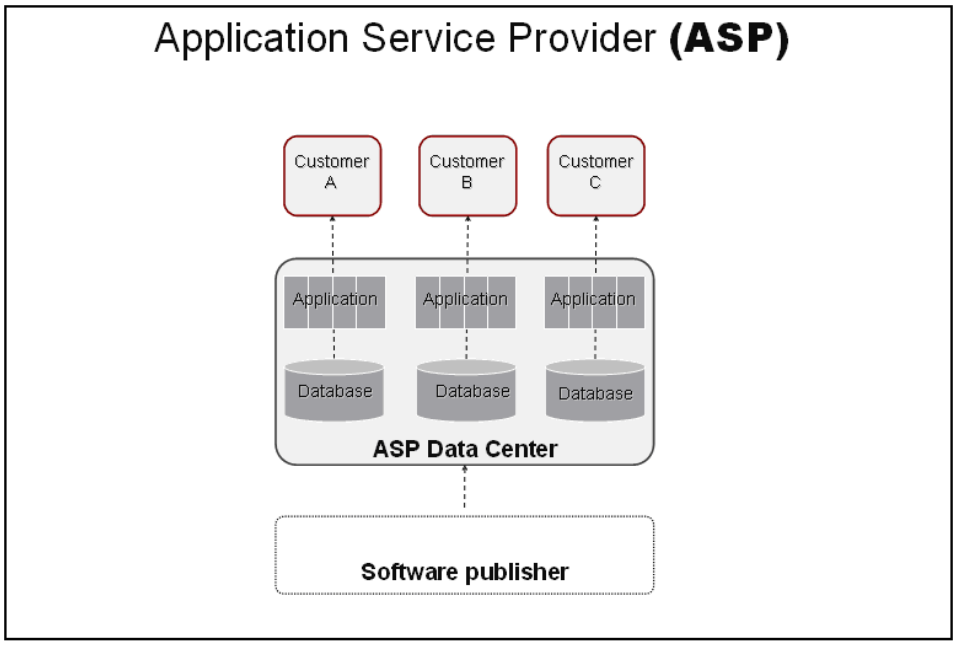


**Figure 2 – ASP Architecture**

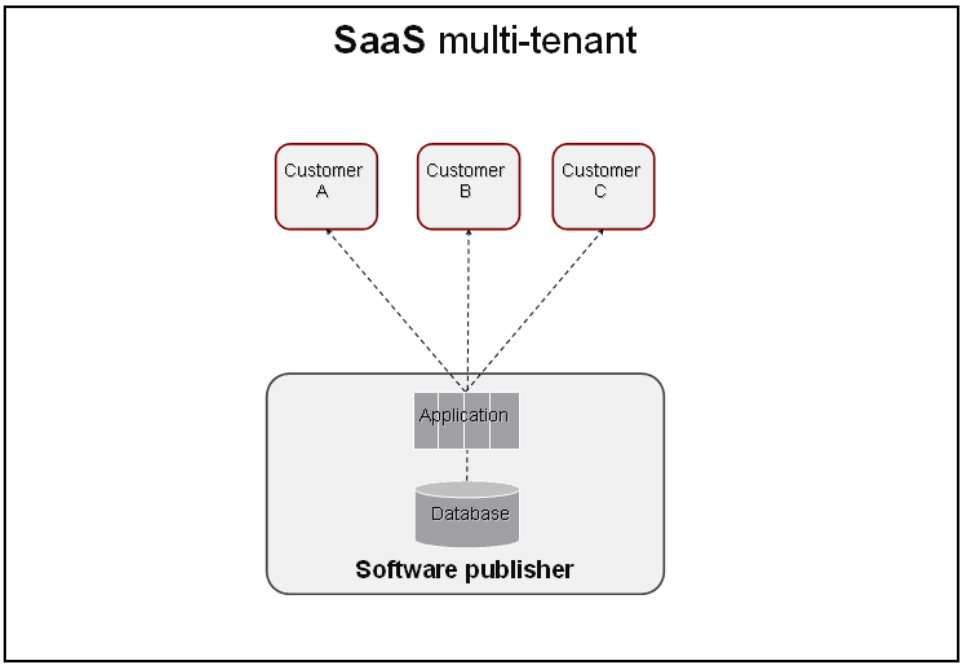


**Figure 3 – SaaS Architecture**

Today, SaaS applications are expected to take advantage of the benefits of centralization through a single-instance, multi-tenant architecture, one-to-many delivery model and to provide a feature-rich experience competitive with comparable on-premise applications. A typical SaaS application is offered either directly by the vendor or by an ambassador called an aggregator, which bundles SaaS offerings from various vendors and offers them as part of a unified application platform.

In contrast to the one-time permanent licensing model commonly used for on-premise software, SaaS application access is frequently sold using a subscription model, with customers paying an continual fee to use the application. Billing and fee structures vary from application to application; some providers charge a straight rate for unlimited access to some or all of the application's features, while others charge different rates that are based on usage.

From technical view, the SaaS provider hosts the application and data centrally — deploying fixes, patches and upgrades to the application transparently, and delivering access to end users over the Internet through a browser or thin client application. Many vendors provide application programming interfaces (API) that make the applications data and functionality reachable to developers for use in creating composite applications. Various security mechanisms can be used to keep sensitive data safe in transmission and storage. Applications providers might provide tools that allow customers to modify the data schema, workflow, and other features of the application's operation for their use [18].

### 2.2.5 SaaS vs. SOA

The main differences are [21]:

- Service Oriented Architecture (SOA) is a way of designing and building software. It is a manufacturing and construction model.
- Software as a Service (SaaS) is a way of receiving software through an external party to your business similar to telephone, water or power utilities. It is a sales and distribution model.

From the objectives and scope perspective, the SOA network model is a list of potential services to be used in a software system being built; but the SaaS network model is a list of possible services to be delivered. From an owner's perspective, SOA implies various found business services to be used in the system; SaaS implies various business services to be provided. Using existing business services could certainly eliminate software design and development expenses.

Note that SaaS doesn't mean that a software system is delivered as only single service. Instead, a software system could be delivered as multiple services — in a sense, parts of the system could be stand-alone services which work with the big service for the entire system.

From a designer's perspective, SOA describes an architectural model which depicts interaction patterns among constituent service components, whereas SaaS describes interaction patterns among basic components that aren't necessarily services. From a builder's perspective, both SOA and SaaS are overlapping and need to identify a technology (such as Web services) to realize the interaction models defined in the information system model [22].

### 2.2.6 SaaS vs. Cloud Computing

Cloud computing is a kind of dynamic computing in which automatically scalable and often virtualized resources are provided as a service over the Internet. Users need not have knowledge of, expertise in, aware of or control over the technology infrastructure in the "cloud" that supports them.
This concept has the following layers [25]:

- infrastructure as a service (IaaS)
- platform as a service (PaaS)
- software as a service (SaaS)

The two terms are different and SaaS is a sub-service and part of cloud computing. Not all cloud applications are SaaS applications, but actually all SaaS applications are in the cloud, and the cloud is rigidly providing the computing power to run those applications, regardless of the type [24].

- SaaS applications may use the cloud computing but they are not the cloud.
- SOA architectures may or may not be delivered as SaaS but they are not instances of SaaS.
- Cloud applications do not need to be delivered as SaaS, although it may be [23].

# CHAPTER 3

# FEATURE MODEL OF SAAS

Software-as-a-Service (SaaS) architecture is encompassed with several domains and technologies. Building a SaaS application requires elaborated analysis of these domains. We propose a feature model to analysis SaaS which expresses the variabilities and commonalities in SaaS and related domains. This feature model defines the valid combinations of features, where each combination corresponds to architecture. This chapter depicts the key capabilities of concepts in SaaS, free of implementation technology details. By this feature model, we aim to help designer to see the scope of the SaaS architecture, comprehend and reason about the alternatives.

In the industrial area, traditional software applications are being antiquated. Now, many vendors are migrating to SaaS. Although this increasing trend of SaaS, there is lack of systematic approach for analyzing and building SaaS architectures. Since any kind of software can be delivered as a service, SaaS application architectures vary in wide range and despite the variety of SaaS applications; there are common architectural components that each system needs to include. In this study, we propose a feature-based analysis method to describe the common and variable characteristics of SaaS for building SaaS architectures.

## 3.1 Domain Analysis

Domain analysis can be defined as the process of identifying, capturing and organizing domain knowledge about the problem domain with the purpose of making it reusable when creating new systems. The UML glossary provides the following definition of the term domain:

Domain: An area of knowledge or activity characterized by a set of concepts and terminology understood by practitioners in that area. A survey of domain analysis methods shows that these methods include the similar kind of activities. Figure 5 represents the common structure of domain analysis methods as it has been derived from survey studies on domain analysis methods.

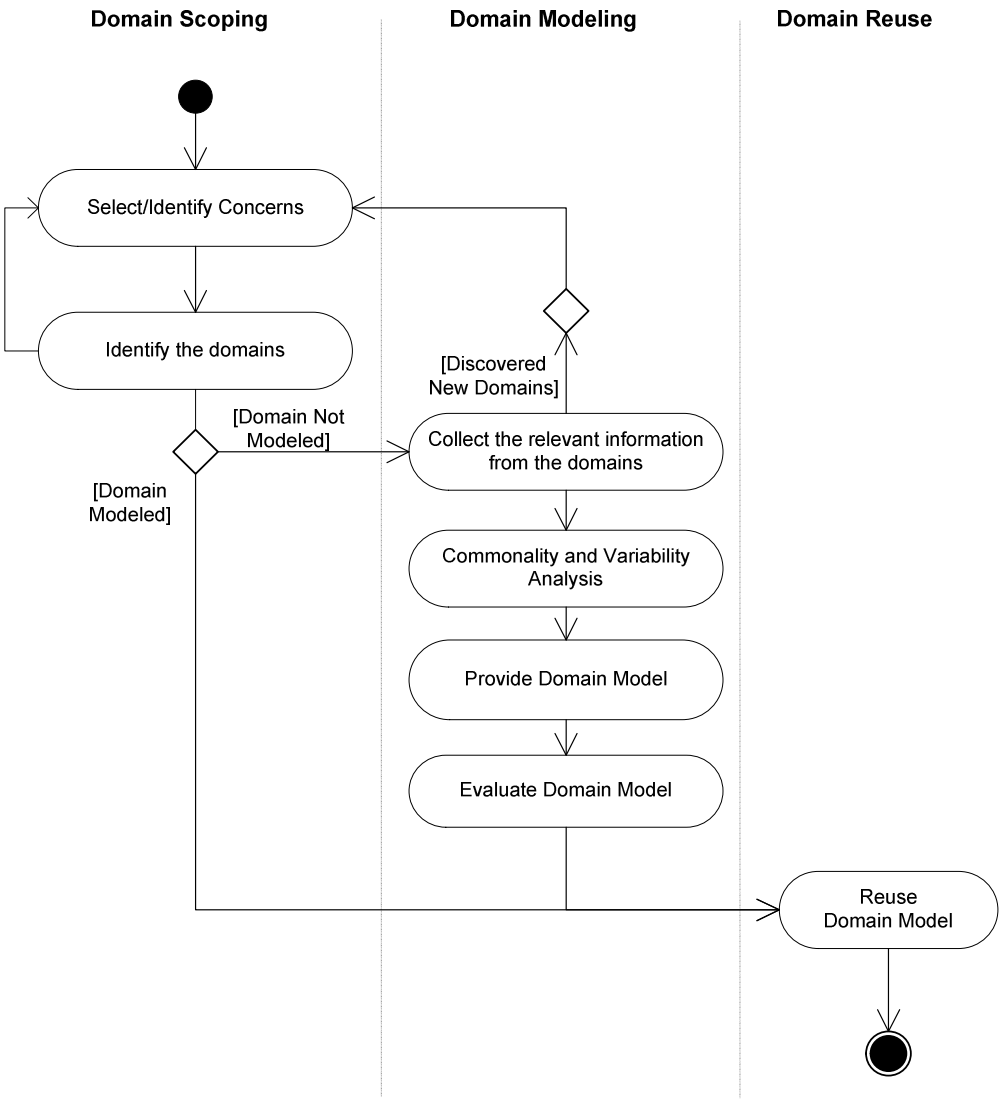


**Figure 4 – Common structure of domain analysis methods**

Conventional domain analysis methods consist generally of the activities Domain Scoping and Domain Modeling: Domain Scoping identifies the domains of interest, the stakeholders, and their goals, and defines the scope of the domain. Domain Modeling is the activity for representing the domain, or the domain model. The domain model can be represented in different forms such as object-oriented language, algebraic specifications, rules, conceptual models etc.

Typically a domain model is formed through a commonality and variability analysis to concepts in the domain. A domain model is used as a basis for engineering components intended for use in multiple applications within the domain.

### 3.1.1 Feature Modelling

Feature models are often used for defining the model of products for a given application domain. Feature modeling has also been extensively used in domain engineering. Hereby, a feature model is a result of a domain analysis process in which the common and variant properties of a domain are elicited and modeled. In addition, the feature model identifies the constraints on the legal combinations of features. A feature model can thus be considered as a specification of the family.
In feature modeling, the common characteristics of software systems of each domain are called commonalities and the particular parts characteristics are called variabilities.

Relationships between a feature and its sub-features are represented in Figure 6:

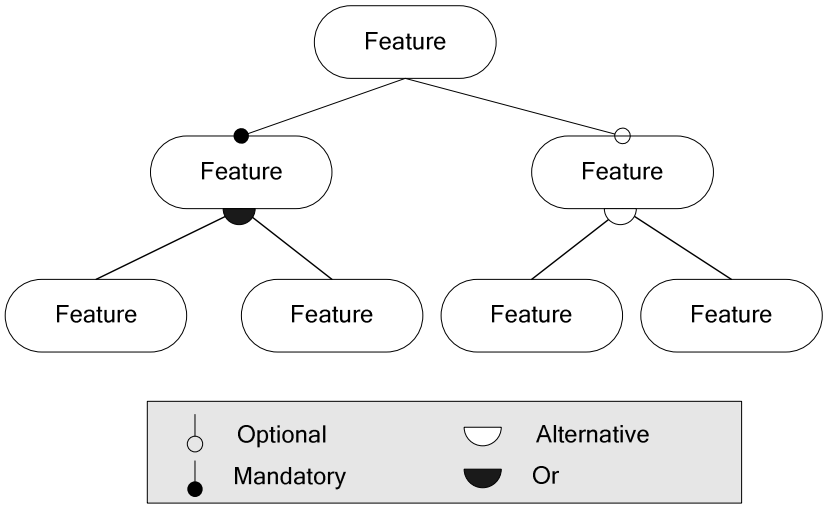


**Figure 5 – Feature Model Diagram**

- Mandatory: child feature is required.
- Optional: child feature is optional.
- Or: at least one of the sub-features must be selected.
- Alternative (xor): one of the sub-features must be selected

In addition to these relationships, cross-tree constraints are possible:

- Requires: The selection of a feature in a product implies the selection of another.
- Excludes: Two features cannot be part of the same product.

## 3.2 Feature Model for SaaS

To derive the feature model of SaaS from the scratch, we need reference architecture of SaaS. Initially, common components and layers should be identified and then the variable parts should be explored. For this reason, we have investigated several SaaS architectures and formed the reference architecture of SaaS.

### 3.2.1 SaaS Reference 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 4. SaaS has a multi-tier architecture with multiple thin clients. In Figure 4 the multiplicity of the client nodes is shown through the asterisk symbol (*). In alignment with the philosophy of SaaS clients do not have lot functionality

installed but rather they rent and access these 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.

We have defined the layers that are provided by the cloud (internet) as Distribution Layer, Presentation Layer, Business Service Layer, Application Service Layer, Data Access Layer, Data Storage Layer and Supporting Service Layer.

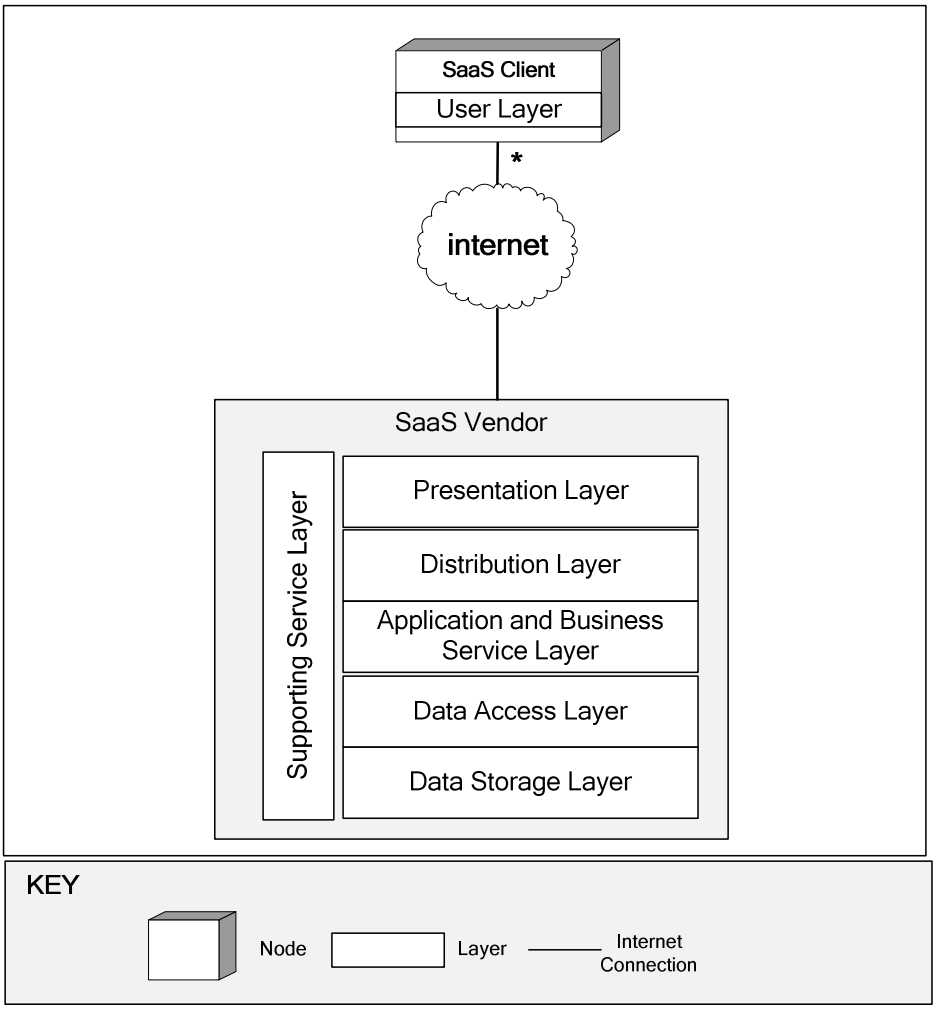


**Figure 6 – SaaS Reference Architecture**

Distribution Layer defines the functionality for load balancing and routing. The Application and Business Service Layer represents services such as identity management, application integration services, and communication services. Presentation Layer handles the displaying data to the users. 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, security, and fault management. Each of these layers can be further decomposed into sub-layers.

### 3.2.2 Top Level Feature Model

By means of the reference architecture of SaaS, we represent the top level feature diagram of SaaS in Figure 7. This diagram merely indicates the layers in SaaS architecture.

In consequence of the SaaS approach of centralized application management, the client side includes one layer, User Layer which usually includes a web browser and/or the functionality to access the web services of the providers. From the perspective of the clients, we have defined the layers that are provided by the cloud (internet) as Distribution Layer, Presentation Layer, Application and Business Service Layer, Data Access Layer, Data Storage Layer and Supporting Service Layer. Distribution Layer defines the functionality for load balancing and routing. Presentation Layer handles the displaying data to the users. 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, and fault management. Each of these layers can be further decomposed into sub-layers.

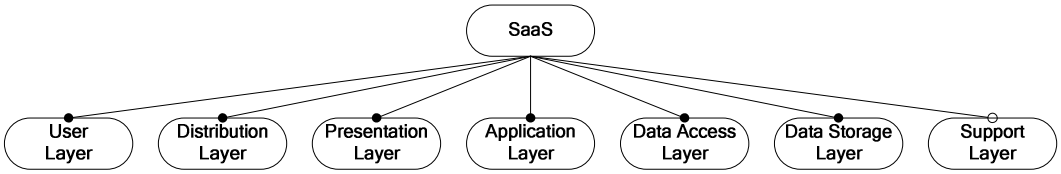


**Figure 7 – Top Level Feature Model**

### 3.2.3 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, so it should provide a usable, easy to navigate and charming interface.

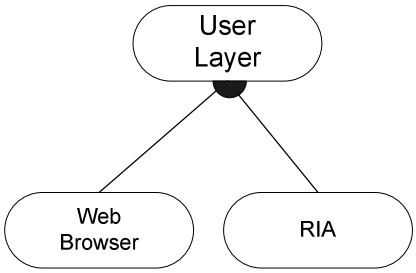


**Figure 8 – User Layer**

*Web Browser:* SaaS clients have to be thin clients. SaaS applications are often accessed by web browsers but browsers are not the only manner to use applications. Any other thin client could be used.

*RIA:* Sometimes RIA is used as a client, especially on mobile platforms. A Rich Internet Application (RIA) is a web application that has all the functionality of a full-fledged desktop program. These programs don't require installation.

### 3.2.4 Distribution Layer

This layer is the interlayer between the internet and the SaaS application. This layer provides scalability, availability and security. Core components of this layer are firewalls and load balancers. Distribution layer is also used for routing the requests to different environments for resiliency or data privacy.

Scalability is one of the most desired features of SaaS applications. SaaS architecture should be able to handle increasing demands. In this layer, scalability is provided by load balancer.

Another important quality attribute is availability. SaaS is changing the way of using software. Software is not owned any more, software is consumed. Clients used to reach their desktop applications anytime. If SaaS is going to replace this use, it has to provide high availability. Single point of failure should be avoided by multiplying load balancers.

*Firewall:* 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.

*Firewall farm:* A firewall farm is a group of connected firewalls. Network traffic is balanced by going through of the firewalls in the farm. So, the firewall farm requires load balancer too. There are two sides of the farm: inside and outside.

Load balancers are connected to both of the sides. In case of huge demands, firewall farm provides security in a scalable manner.

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

*Type*: Load balancers are grouped in two types; hardware based and software based load balancers.

*Hardware Load Balancer:* A dedicated hardware designed for load balancing. This kind of load balancers has better performance and more robust according to software based one, but it is much more but more costly and more complex to configure.

*Software Load Balancer:* Standard server hardware and standard operating system used as a load balancer. Due to developing technology, typical servers can provide enough performance.

*Strategy:* Load balancing strategies decide how to distribute requests to target devices. These are grouped in two strategies. We are going to mention about the common strategies though there exists much more of them [75] [76].

*Passive Strategies:* Load balancers use already defined strategies regardless the run time conditions of the environment.

*Round Robin:* One of the most use load balancing strategy. Requests are sent to server turn by turn.

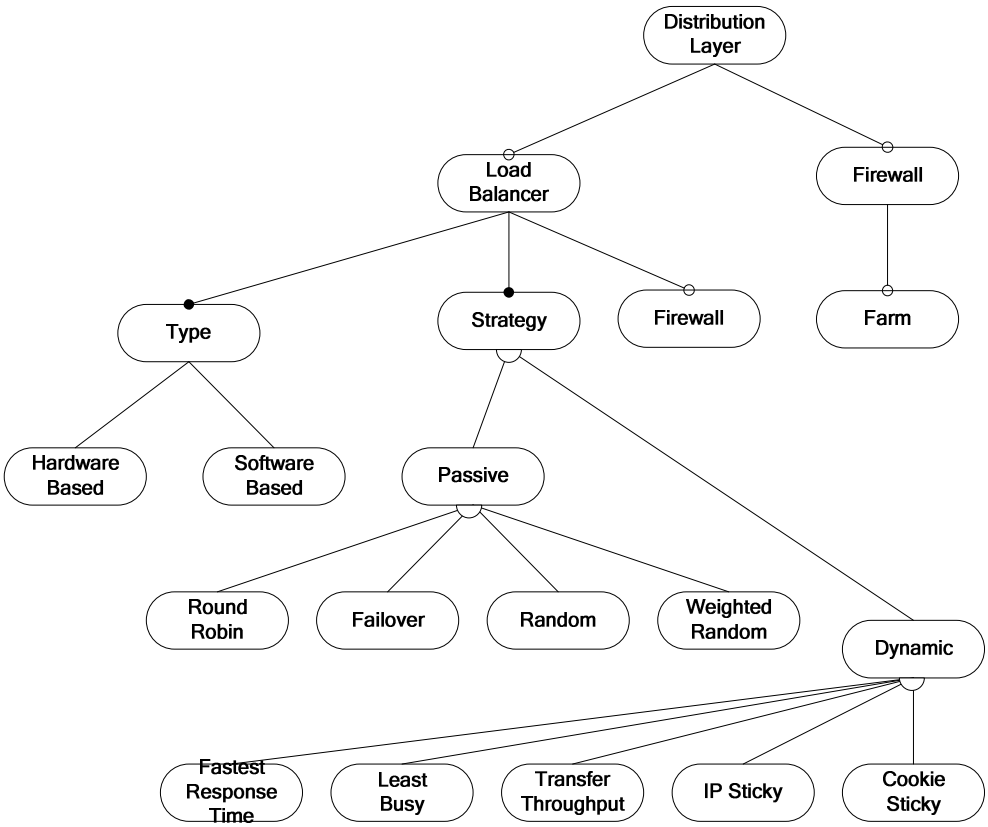


**Figure 9 – Distribution Layer**

*Failover:* In this case, there is a list of load balancing policy ordered by priority. Load balancer chooses the most significant one, and then the next if the previous policy is disabled or inaccessible.

*Random:* Target machines are assigned randomly for each request.

*Weighted Random:* Target machines are chosen randomly. Weight of the targets can be configured by a policy. This method is useful if the load balancer does not know the actual performance of the server.

*Dynamic Strategies:* Load balancers are aware of information of the targets and routes the requests based on traffic patterns according to selected strategy.

*Fastest Response Time:* Statistics are measured and fastest responding server is selected.

*Least Busy:* Load balancer tracks the number of connections of the servers and sends the request to the least busy server.

*Transfer Throughput:* Server throughput performance (average throughput) is measured and the requests are sent to the server with the highest performance.

*IP Sticky:* This method will simply direct the same user back to the same web server based on their IP address.

*Cookie Sticky:* The client is issued a cookie on the first connection with the internal server ID. Subsequent requests will read the cookie and redirect the request to the appropriate web server.

Dynamic strategies distribute the load fairer where passive strategies consume less time.

### 3.2.5 Presentation Layer

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.

*Web Server:* A web server handles HTTP requests. 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.

SaaS applications are interactive applications, not static web pages but contain static content. Web servers are worth for scalability, fault tolerance and performance issues.

*Web Proxy Server:* To increase the performance of the web servers and presentation layer, caching web contents and reducing load is performed by web proxy servers.

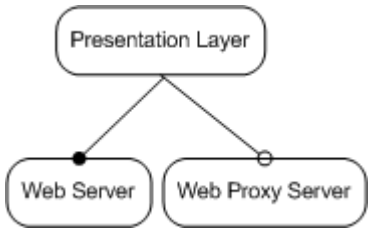


**Figure 10 – Presentation Layer**

By the way, one of the features of SaaS is mobility. Users can access the SaaS applications anytime from anywhere by just using a thin client. Popularity of

mobile platforms increases and habits of internet usage are changing. Accessing same application by different platforms may cause some presentation problems: displaying a web page for a limited sized mobile phone is not same with displaying on a wide screen monitor. Web proxy servers can be used for reformatting the presentation for special purposes as well.

### 3.2.6 Application Layer

Application layer 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.

SOA is a software construction model by development and integration of services. Various systems can communicate interoperable. SOA helps to create reusable, modular systems.

SaaS leverages SOA and adjusting the architecture for SaaS needs adding components that is loosely coupled.

SaaS and SOA are neither alternatives of each other nor replaceable technologies. Further, they have overlapping concepts and they are, in some way, complementary technologies.

SaaS applications based on SOA are more flexible, scalable and reusable from the nature of service orientation. SOA also enables SaaS applications to better integrate. SaaS architecture which has SOA capabilities will benefit from these advantages. Indeed, SOA deployments which use SaaS will have more return of

investment. So, using both technologies will improve the advantages of each other.

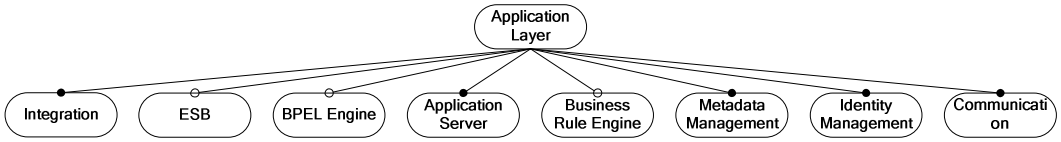


**Figure 11 – Application Layer**

### 3.2.6.1 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 is simply 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 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.

*Server Clustering:* 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 [78].

*Asymmetric Clustering:* In asymmetric clusters, a standby server exists to take control in case of another server gets of failure. Asymmetric clustering is generally used to for high availability and scalability for read/write operations such as databases, messaging systems, and file and print services. While the system runs, there can be a planned downtime such as maintenance or unplanned downtime such as failure. If one of the nodes in a cluster becomes unavailable, another node takes the responsibility of the failed node. The standby server has no other capabilities according to the primary server and it is not more capable or does more useful work than the primary server. A less capable and less expensive standby server is often used when primary servers are configured for high availability and fault tolerance with multiple redundant subsystems [78].

*Symmetric Clustering:* In symmetric clusters, every server in the cluster do actual job. That is to say, each server is the primary server for a specific task. If one server fails, the remaining server continues to process its assigned task as well as the applications on the failed server. Symmetric clusters are more cost-effective because they use the cluster's resources efficiently. However, in there exists a failure, other servers take extra load and this case increases the risk of rest to fail [78].

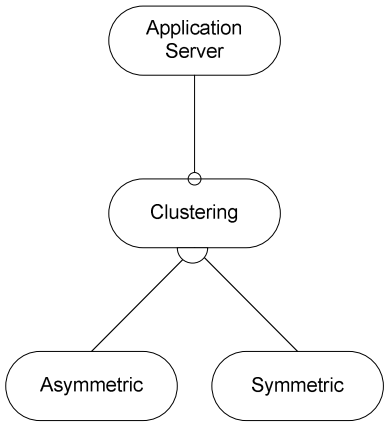


**Figure 12 – Application Server**

#### 3.2.6.2 ESB

Formerly, software systems used to be standalone and independent. In today's conditions, enterprise applications need much more information exchange, collaboration and integration. Integrating the application is provided by an infrastructure. Two main architectures for this infrastructure are Hub-and-Spoke and Bus architectures.

Hub & Spoke is more centralized architecture, simple, easy to implement but it does not scale well. On the other hand Bus architecture is more distributed, it has pluggable components, scales better but it is more complex.

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 is a platform for integrating services and provides enterprise messaging system. We can think ESB as a variation of bus based enterprise application integration for services. Using an ESB system does not mean implementing a service oriented architecture but they are highly related and ESB facilitates SOA.

#### 3.2.6.3 BPEL Engine

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 an orchestration language that defines business processes.

Some simple tasks may be performed by ESB but more complex business processes should be defined by BPEL. To interpret and execute BPEL there is a need of an engine.

#### 3.2.6.4 Metadata Management

SaaS has single instance, multi-tenant architecture. Sharing the same instance to many customers brings the problem of customization. In SaaS architecture, customization is done by 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 the "*Storage Layer.Metada Repository*" feature.

#### 3.2.6.5 Business Rules 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.

#### 3.2.6.6 Integration

Clients used to use on premise applications and store their data in their own side. By SaaS, all the control, upgrade, and maintenance are handled by SaaS vendors. As seen, SaaS brings simplicity and ease of use. However, data integration is a challenging part of SaaS. SaaS applications need to use customer data which resides at the customer's side. On the other hand, the customer 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 format. Manipulating data will affect 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.

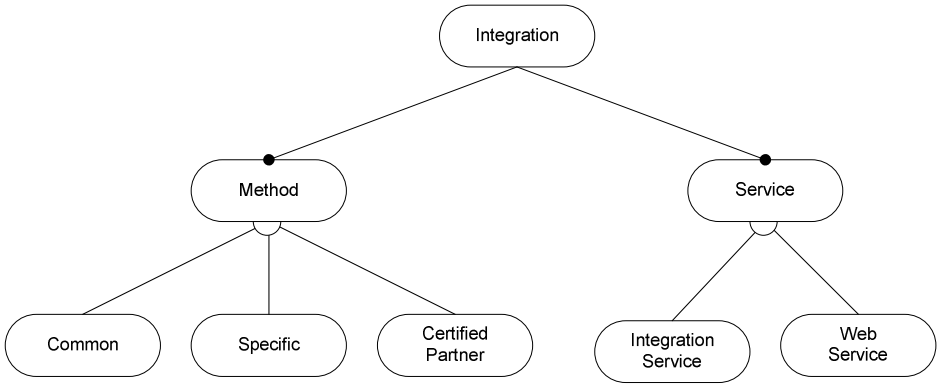


**Figure 13 – Integration**

There are three approaches for providing integration: common, specific and having a certified partner.

*Common Integration:* This approach provides services for all customers. This feature requires "*Integration.Services.Web Services*" feature.

*Specific Integration:* In this approach, services are customized for each customer. This feature requires "*Integration.Services.Integration Services*" feature.

*Certified Partner:* SaaS vendor delegates the integration to another vendor which is a specialist for SaaS integration. SaaS vendor still needs to provide web services, but leaves the control to the experts and focuses on its application. This feature also requires "*Integration.Services.Web Services*" feature.

*Integration Service:* In this approach, 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.

*Web Service:* SaaS vendor provides a Standard approach for customers as web services. Customers take responsibility for SaaS integration. Comparing with the first approach, customers have to do much more and have IT experience. But this is more scalable solution for vendors.

#### 3.2.6.7 Identity Management

Security is an important issue for SaaS environment. Encrypted communication is not only thing to deal with security. While exposing services, vendors have to know who is accessing to the services, what about the credentials and authentication about those consumers. Account management, user access control, identification, password management is part of the protection. These concepts are all covered by SaaS systems; hence SaaS architecture has to involve an identity management system.

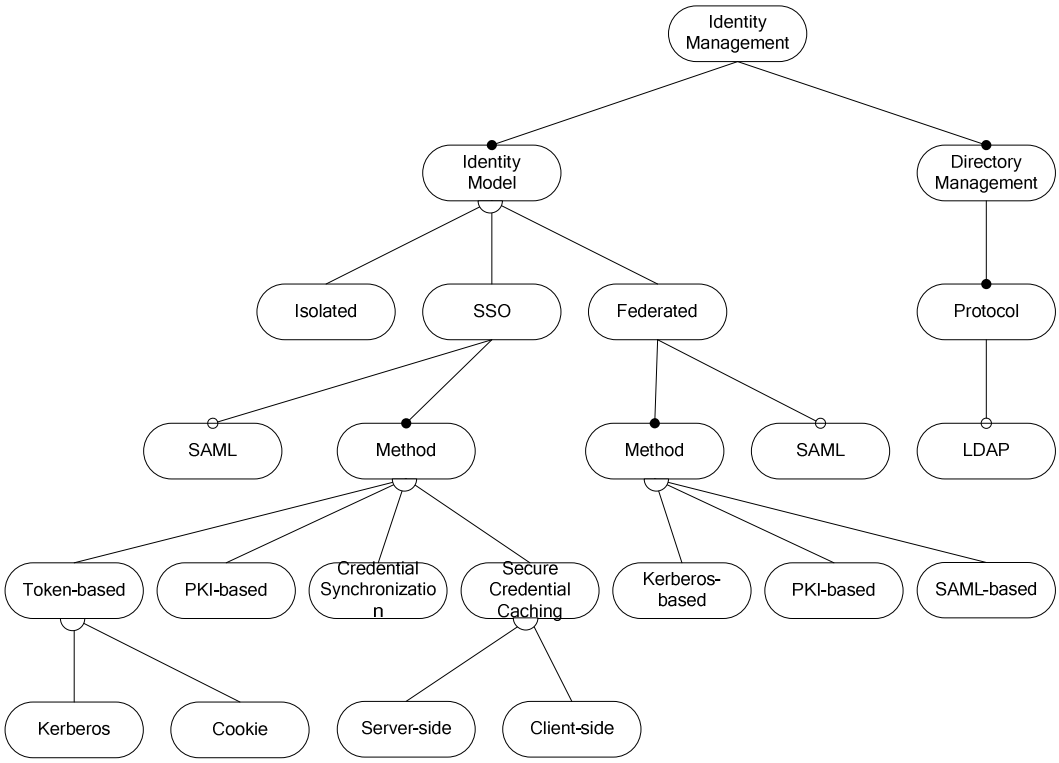


**Figure 14 – Identity Management**

Identity management systems, designed for cost effectiveness may be poor for usability. In SaaS architecture, the identity management has to be designed also regarding to ease of use (e.g. Single Sign-On) and scalability.

*Isolated Identity Management:* The most common and simplest identity management model is isolated one. Each service provider associates an identity for each customer. Opposing to simplicity, it is an unmanageable model in case of the growth of number of users and users regret to remember passwords and login to their accounts for each service.

This model could be used if the vendor does not have too many users and does not need interact with other services much.

*SAML:* Security Assertion Markup Language is the XML based security standard created to enable portable identities and the assertion of these identities. In enables different services to exchange authentication and authorization credentials. Because it is XML based, SAML is platform-independent. SAML provides secure identity communication for SSO, distributed and federated identity management.

*Single Sign-On:* Single Sign-On is a centralized identity management model, which allows users to access different systems using a single user ID and password. SSO systems have Authentication Server or Identity Provider to authenticate users. Once a user is identified, the user is allowed access disparate systems within an organization. So the users do not have to keep several user identities and passwords and do not have to log in for each service [62] [77].

SSO identity management model has the alternative architectures listed below:

- Token-based
    - Kerberos
    - Cookie
- PKI-based
- Credential Synchronization
- Secure Client-side Credential Caching
- Secure Server-side Credential Caching

*Token-based:* In a token-based SSO architecture users get a temporary software token when they have been successfully identified and authenticated. This token can be cached on the user's machine (client-side) and can be reused when another authentication domain requests. Token validation uses cryptographic methods that are based on secret keys among the Trusted Third Parties (TTP), which represents a trust relationship between primary and secondary authentication domains.

*Kerberos:* In the Kerberos model users are authenticated the Kerberos Key Distribution Center (KDC) which is a central authentication service. They receive a software token, named ticket, if their authentication credentials are valid.. The tickets that the user has been authenticated successfully by a trusted authentication service – the Kerberos KDC. The Kerberos token-based SSO generally uses Remote Procedure Calls (RPCs) to transport authentication tickets.

*Cookie:* In the environments which use HTTP, token-based SSO can be provided using HTTP cookies. Cookie is a token which the browser uses to realize that the user has authenticated to some application. The value of the cookie is used as the SSO key, often a session ID generated by the application.

*PKI-based:* In a Public Key Infrastructure-based (PKI-based) SSO architecture users initially register themselves at a trusted authentication authority (certification authority, CA) or at one of the authentication authority's registration agents (registration authorities, RA). During this registration process different events occur: users identify themselves by using a set of credentials; an asymmetric key pair is generated; and finally the public key of this key pair is offered to the CA or RA for certification. Finally user's credential is verified according to the user's credentials and the public key. If the credentials are validated successfully it will generate a public key certificate and send it back to the user. The user's public key certificate and then the user's private key are cached on the user's machine. They both are used to generate software tokens which is similar to the ones used in token-based SSO systems. These tokens are used to verify the user's identity to other authentication authorities in subsequent requests [62].

*Credential Synchronization:* Credential Synchronization is a system which synchronizes user passwords between the different authentication centers. Despite

the support of multiple credentials for each user, these users are distinguished by credential synchronization mechanism. Credential synchronization systems usually use a single master credential database. This database can be used by administrators to make the user credentials up-to-date [62].

*Secure Credential Caching:* In this type of Single Sign-On mechanism, credentials are cached either in server or client side. Later, these credentials are served to other authentication centers on demand [62].

*Secure Client-side Credential Caching:* In this model, a set of primary credentials are used to unlock a user's credential cache. Then, if the user wants to access resources requiring different authentication credentials, the local credential cache provides credentials automatically to the authentication authorities. If the credentials are valid, then the user will be logged on to the other resource servers [62].

*Secure Server-side Credential Caching:* Opposite to the use of a secure client-side cache, secure server-side credential caching SSO architectures keep the credentials in a central repository on the server-side. In contrast to a credential synchronization-based SSO architecture, the credentials used in a secure server-side credential caching SSO architecture are not consequently the same for every authentication authority [62].

The comparison of the SSO architecture is as below [62]:

**Table 1 – Advantages and Disadvantages of different SSO Architectures**

| | **Advantages** | **Disadvantages** |
|---|---|---|

| | | |
|---|---|---|
| Token-based | • Single set of credentials simplifies life of user and administrator<br>• Software usually comes bundled with OS software | • Requires a homogeneous authentication infrastructure environment<br>• Relies on symmetric cryptography |
| PKI-based | • Single set of credentials simplifies life of user and administrator<br>• Software usually comes bundled with state-of-the-art OS software<br>• Relies on asymmetric cryptography | • Can only deal with a single set of credentials<br>• Complex certificate validation logic. Requires a lot of processing on the client side<br>• Requires a homogeneous authentication infrastructure environment (all services and applications must be PKI-enabled) |
| Credential Synchronization | • Can deal with many different credentials<br>• Does not require a homogeneous authentication infrastructure environment<br>• Does not impact the client-side (no extra software needed) | • Credentials are kept identical on different platforms<br>• Does not provide true SSO (unless it is combined with a secure client-side caching mechanism)<br>• "Key to the kingdom" |

| | | |
|---|---|---|
| | | argument<br>• Multiple sets of credentials complicate life of user and administrator<br>• Requires extra software on server infrastructure-side |
| Secure Client-side Credential Caching | • Can deal with many different credentials<br>• Does not require a homogeneous authentication infrastructure environment<br>• Requires a "secure" client-side credential cache – it is not recommended to use it from portable client devices or OSs with a bad security reputation | • Multiple sets of credentials complicate life of user and administrator<br>• Has important impact on client-side (requires extra software or state-of-the-art OS) |
| Secure Server-side Credential Caching | • Can deal with many different credentials<br>• Does not require a homogeneous authentication infrastructure environment | • Requires a credential synchronization mechanism (may be part of the SSO product)<br>• Multiple sets of credentials complicate life of user and administrator |

| | | |
|---|---|---|
| | | • Requires extra software on server infrastructure-side<br>• Has impact on client-side (requires extra software) |

As a result, secure client-side credential caching approach is desired and well-suit for SaaS architectures because this approaches requires additional software on client side which is contrary to SaaS philosophy; but there are no strict limitations not to select this SSO method.

*Federated Identity Model:* SSO has advantages both for users and providers. SSO makes the identity management of systems easy inside an organization. Different organizations can work together, be partners of each other. Leveraging a single authentication system is a demanded property. Federated identity is very close to SSO, does the same with SSO, but across different organizations. This model provides the same benefits with SSO: scalability, ease of use, ease of management. There are three most used approaches [62] [77]:

- Kerberos-based Federation
- PKI-based Federation
- SAML-based Federation

*Kerberos-based Federation:* In a federated authentication infrastructure a foreign TTP validates the foreign credentials and those credentials are also accepted by an organization's proper authentication authority. The justification why they're accepted is because there's an agreement, trust or federation between the foreign and a company's proper authentication authorities. This form does not require a

copy of the foreign organization's credential DB. Also in this case the users only have to devote effort to a single set of credentials.

*PKI-based Federation:* PKIs use the notion of certification authority hierarchies, cross-certification, bridge certification authorities or any other CA-to-CA interoperability between certification authorities solution to set up federations.

*SAML-based Federation:* SAML is platform-independent, because it is based on XML. SAML is also authentication method neutral: for example, it could be used to create federations between PKI and Kerberos-based authentication infrastructures.

The comparison of the federation mechanisms is as below [62]:

**Table 2 – Comparing Federation Mechanisms**

| | **Kerberos-based federation** | **PKI-based federation** | **SAML-based federation** |
|---|---|---|---|
| **Authentication Technology** | Kerberos | PKI | Any |
| **Platform support** | Many | Many | Many |
| **Support for entity authentication** | Yes | Yes | Yes |
| **Support for data authentication** | No | Yes | Under development |

| | | | |
|---|---|---|---|
| **Authorization Federation Support** | Yes, but not standardized | Yes, but very few products support it | Yes |
| **Granularity of trust relationship and security policy support** | Very Monolithic, no policy support | Support for granular trust and security policies in some products | Under development |
| **Status** | Standardized | Standardized, though standardization is not complete | Under development |

*Directory Service:* To manage the identity data, a directory service is an essential component of identity management. A directory service is the software system that stores, organizes and provides access to information in a directory. This feature requires "*Storage Layer.Directory Server*" feature to store directory.

*LDAP:* Lightweight Directory Access Protocol is an Internet protocol that programs use to look up information from a server. LDAP is used for querying and modifying data using directory services.

#### 3.2.6.8 Communication

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 manner for communication is e-mailing but some other protocols can be used as an alternative.

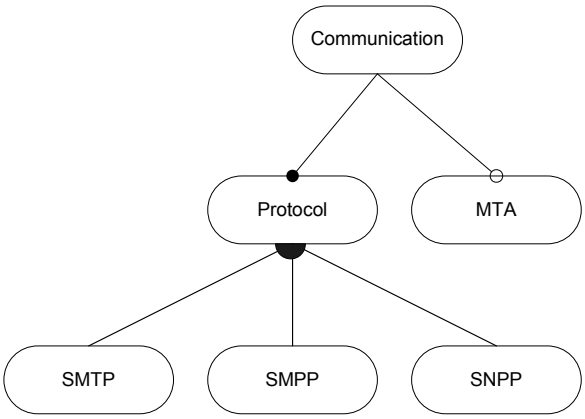


**Figure 15 – Communication**

*MTA:* Mail Transfer Agent is a software system which transfers mail between computers. MTA can accept or forward messages from other servers. MTA uses SMTP protocol (This feature requires "*Communication.Protocol.SMTP*" feature).

*SMTP:* Simple Mail Transfer Protocol is an Internet standard for electronic mail (e-mail) transmission across Internet Protocol (IP) networks.

*SMPP:* The Short Message Peer-to-Peer protocol is a telecommunications industry protocol for exchanging SMS messages between SMS peer entities.

*SNPP:* Simple Network Paging Protocol is a protocol that defines a method by which a pager can receive a message over the Internet.

### 3.2.7 Data Access Layer

Application (business) layer needs access to data databases. This layer provides facilities for connections and transactions management.

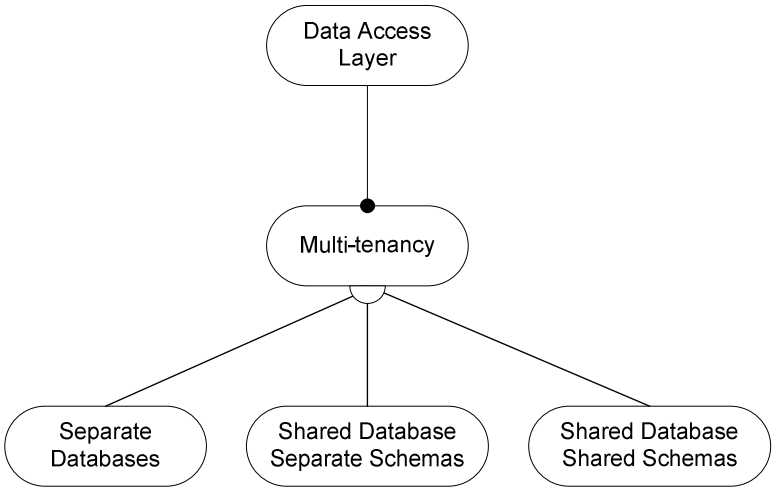


**Figure 16 – Data Access Layer**

A database management system consists of software which manages data (database manager or database engine), structured artifact (database) and metadata (schema, tables, constraints etc.). In our reference architectural model, there are two layers related to data: data access layer and storage layer. Thus, we include database engine to data access layer and database to storage layer. Since this is a layered model, these layers (so the database and database engine) can lie on the same tier at deployment time.

#### 3.2.7.1 Multi-Tenancy

The most important SaaS feature is multi-tenancy. SaaS applications are being used in wide range and getting more popularity because of the ease of use, cost

effectiveness and scalability. These capabilities are due to multi tenant architecture. 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.

*Separate Databases:* Each tenant has its own data set which is logically isolated from others. The simplest way to data isolation is storing tenant data in separate database servers. The facilities of this approach are customizing data model based on user needs is easy, backup and restore operations are simple, data is physically isolated for each tenant, best for scalability, high performance and security. The most important disadvantage is the high cost for maintenance and availability. This feature is suitable for the SaaS providers which have customers that can afford for more security and customizability issues, such as the customers in fields like banking or medical records management.

*Shared Database, Separate Schemas:* In this approach, single database server is used for all tenants. To provide the data isolation, when a user is subscribed to the SaaS system, a discrete set of table spaces (schema) is created. By this approach, data model is still customizable and data isolation is reached by logical separation although tenants' data reside in the same physical server (not a complete separation as separate databases approach). The disadvantage of this approach is incomplete restore is difficult to achieve. Fortunately, soma new technologies are available to overcome this problem. Using the same database server for tenants is more effective and this situation increases the costs. This approach can be used in case of the SaaS tenants do not have hard, restricted rules for some security issues

like co-locating their data with others.

*Shared Database, Shared Schema:* This approach involves using one database and one schema for each tenants' data. A table can contain multiple tenants' data in any order. The tables have additional columns, tenant identifier column, to distinguish the tenants. Restoring operation is similar to shared database, separate schema approach. This approach has the lowest hardware and backup costs. The database server can serve to a large number of tenants by effective usage of capacity. A SaaS vendor can choose this approach if the customers desire to surrender data isolation in exchange for the lower costs.

### 3.2.8 Data Storage Layer

Since all information needs to be stored, this layer is used for the physical storage of data. The persistent data is mainly grouped in three types. One of them is tenants' data related to application. This is the main concern of the data management. Next, metadata is stored in storage layer. This is usually xml formatted files for providing user customization. Lastly, directory server is located in this layer which stores a map of user information for identity management.

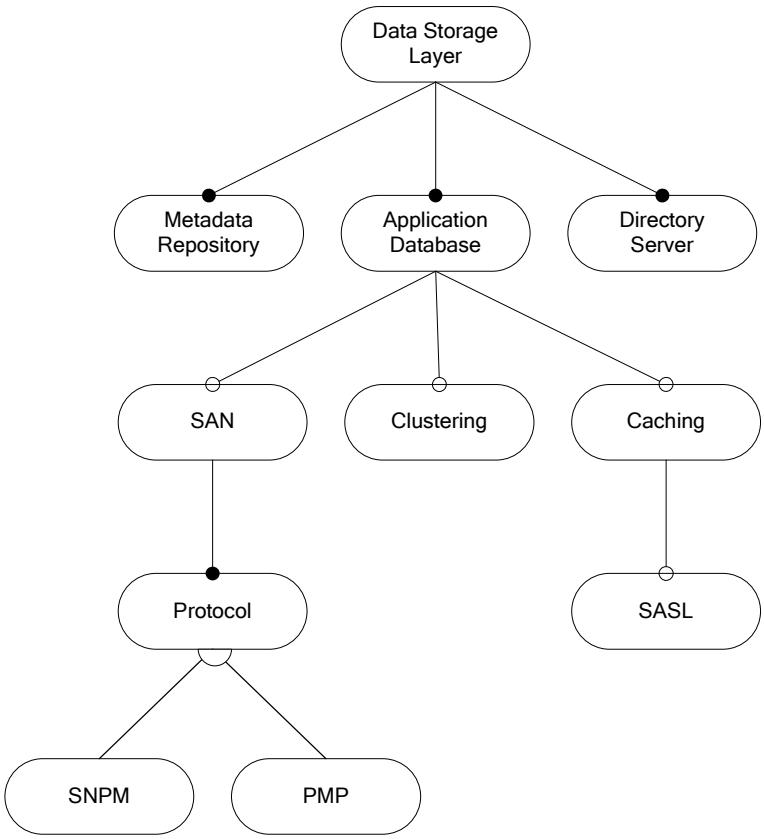


**Figure 17 – Data Storage Layer**

*SAN:* A storage area network is a high-performance subnet, based on fiber channel, whose initial purpose is to move data between heterogeneous servers and storage resources. SAN can be thought as a shared storage bus. Since SAN is a dedicated network based on high performance I/O channel, traffic overhead is avoided.

SAN provides faster and easier data access according to traditional approaches. The primary benefits of a SAN are scalability, performance, availability, reliability and manageability.

SAN uses two protocols: SNMP and Proprietary Management Protocol, the latter enables more security.

*Caching:* Database caching an effective approach to achieve high scalability and performance. Caching means that putting frequently accessed objects into an area to get them faster and easier. This area is usually the memory. By caching disk access and computation are reduced while the response time is decreased. The other benefits of database caching are about scalability, flexibility, availability and performance. In a database, there are three basic types of caching: query results, query plans, and relations.

*SASL:* Simple Authentication and Security Layer, a method for adding authentication support to connection-based protocols. To use SASL, a protocol includes a command for identifying and authenticating a user to a server and for optionally negotiating protection of subsequent protocol interactions. If its use is negotiated, a security layer is inserted between the protocol and the connection [74].

Memcached is a distributed memory object caching system. It is often used to speed up dynamic database-driven websites by caching data and objects in RAM to reduce the number of times an external data source must be read. Memcached is almost being a standard for memory caching. Today huge SaaS applications use memcahced for scalability and performance issues (e.g. Facebook). Memcached may include SASL (Simple Authentication and Security Layer) framework to increase security.

*Clustering:* 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.

*Metadata Repository:* Metadata is stored xml format. Metadata files can be stored either in a database or in a file based repository.

*Directory server:* Directory server 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.

### 3.2.9 Supporting Service Layer

Supporting Service Layer is a cross-cutting layer that provides services for all layers.

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.

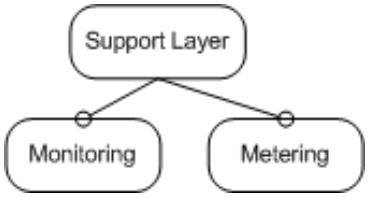


**Figure 18 – Supporting Service Layer**

# CHAPTER 4

# DERIVING SAAS APPLICATION ARCHITECTURE

This chapter describes the approach for deriving various SaaS application architectures, based on the feature model of SaaS.

Although Figure 4 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. For example, Figure 19 shows an alternative application architecture design that is derived from the reference architecture in Figure 4. The design supports the need for multi-tenancy by adopting a single database management system with a shared database and shared schemas for the tenants.

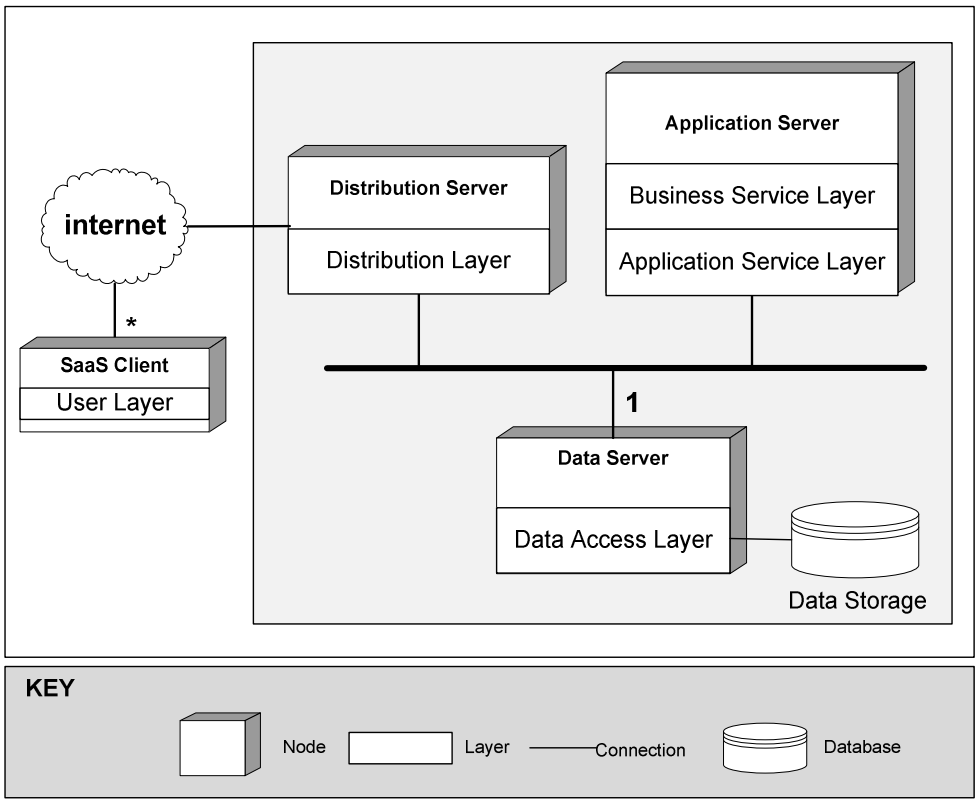


**Figure 19 – SaaS Application Architecture Alternative with Shared Data Servers and separated Single Distribution and Single Application Server**

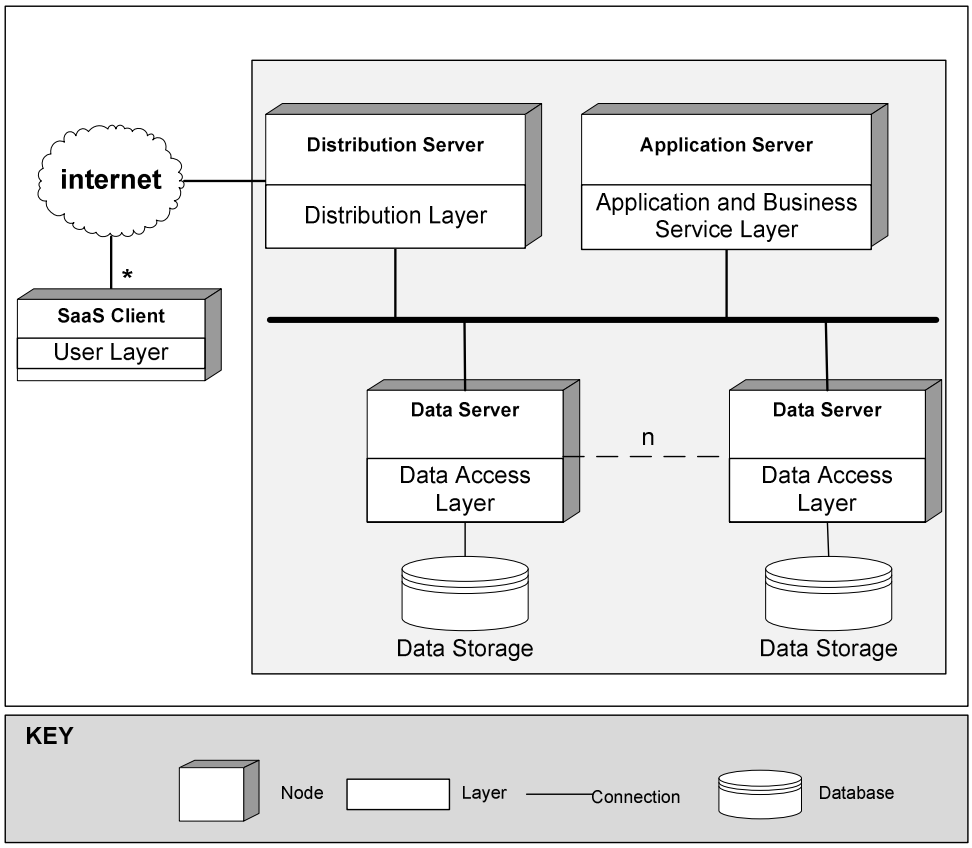


**Figure 20 – SaaS Application Architecture Alternative with Separate Data Servers for Tenants and separated Single Distribution and Single Application Server**

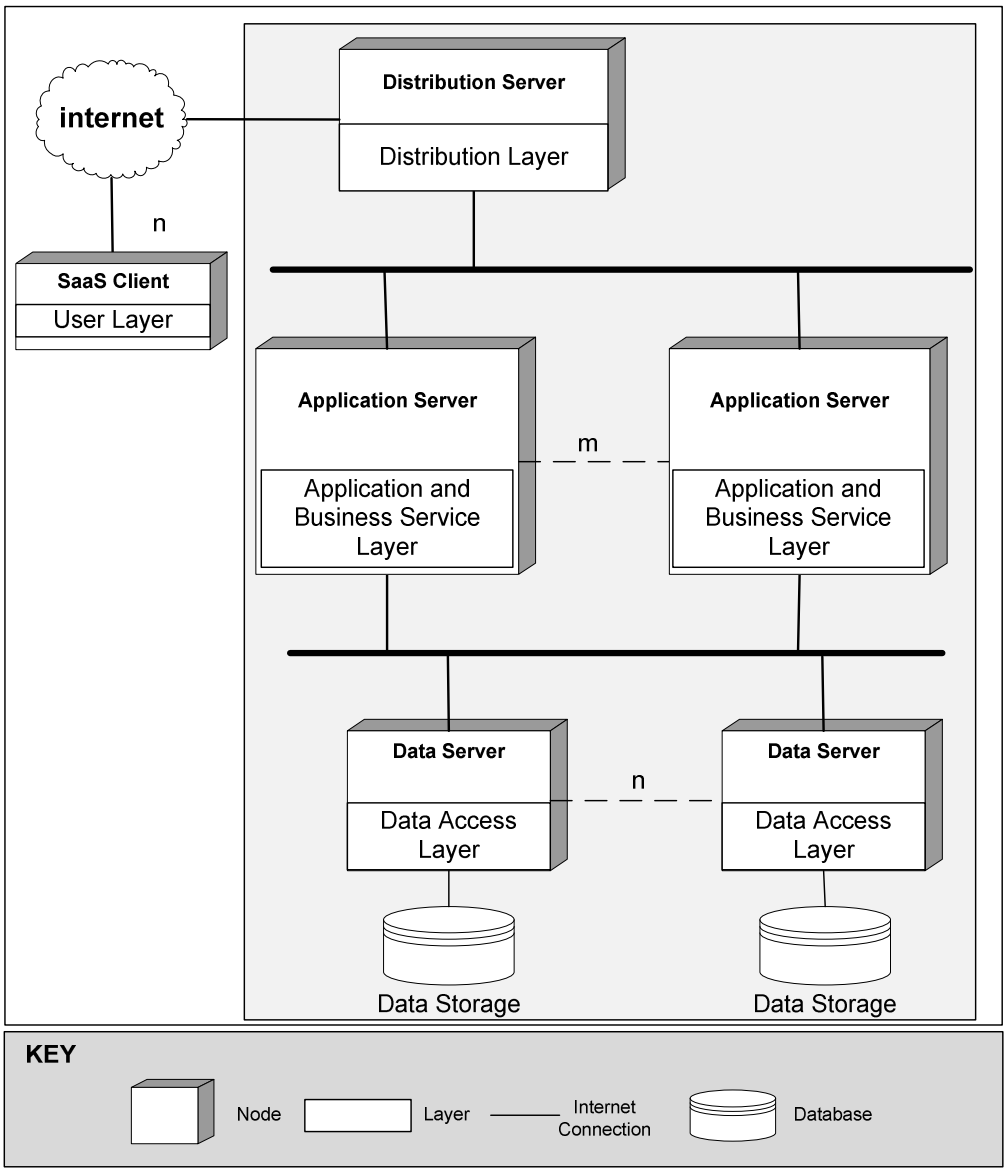


**Figure 21 – SaaS Application Architecture Alternative with Separate Data Servers for Tenants, Separate Application Server, and one Distribution Server**

Figure 20 shows an application architecture in which data storage is not shared but a separate Data Server provided for each tenant. Yet another design alternative is depicted in Figure 21 which shows a more complicated architecture in which

Application Servers are also distributed to m number of multiple nodes to increase performance for multiple tenants.

Obviously these three design models are not the only alternatives and a considerable number of other design alternatives may be derived from the same reference SaaS architecture. Each of these alternatives will be required for different requirements and constraints. Typical requirements that have an impact on the boundaries are the processing power need, different use of I/O, different configuration constraints, etc. All these requirements will not only impact the overall architecture but also have a direct impact on how each layer is designed individually. The architect will have to face with many questions: What kind of distribution method will be selected? Should clustering be used for the application server? Which protocol will be used for communication? What will be the Identity Management System for the system? How will the integration be made? How many tiers should be used?

SaaS application designers must be able to explicitly compare, evaluate and decide between various alternatives based on the relative importance of the requirements and the constraints. Unfortunately, a systematic approach for depicting the space of possible application architectures and the selection of these alternatives is missing. True, while designing SaaS architectures, software engineers apply their knowledge, experience and intuition to compare the design alternatives. However, this process is primarily implicit and lacks explicit support. Without knowledge of the design space it is difficult to specify, compare and select the feasible application design alternative. As such we think that current SaaS methods should provide explicit means to determine and reason about the design space and the individual application design alternatives of SaaS.

## 4.1 Approach for Deriving SaaS Application Architecture

In this section we provide an approach for depicting the design space of application architectures and for selecting the appropriate design alternatives from this design space. The overall process is shown in Figure 22.

The process consists of two basic activities: Family Modeling and Application Modeling. In Family Modeling we define the reference models for SaaS including SaaS Reference Architecture, SaaS Family Feature Model, and SaaS Reference Design Rules. In Application Modeling, based on the reference models, we define the application models including Application Feature Model, Application Design Rules and Application Architecture. We explain the important steps in the following sections.

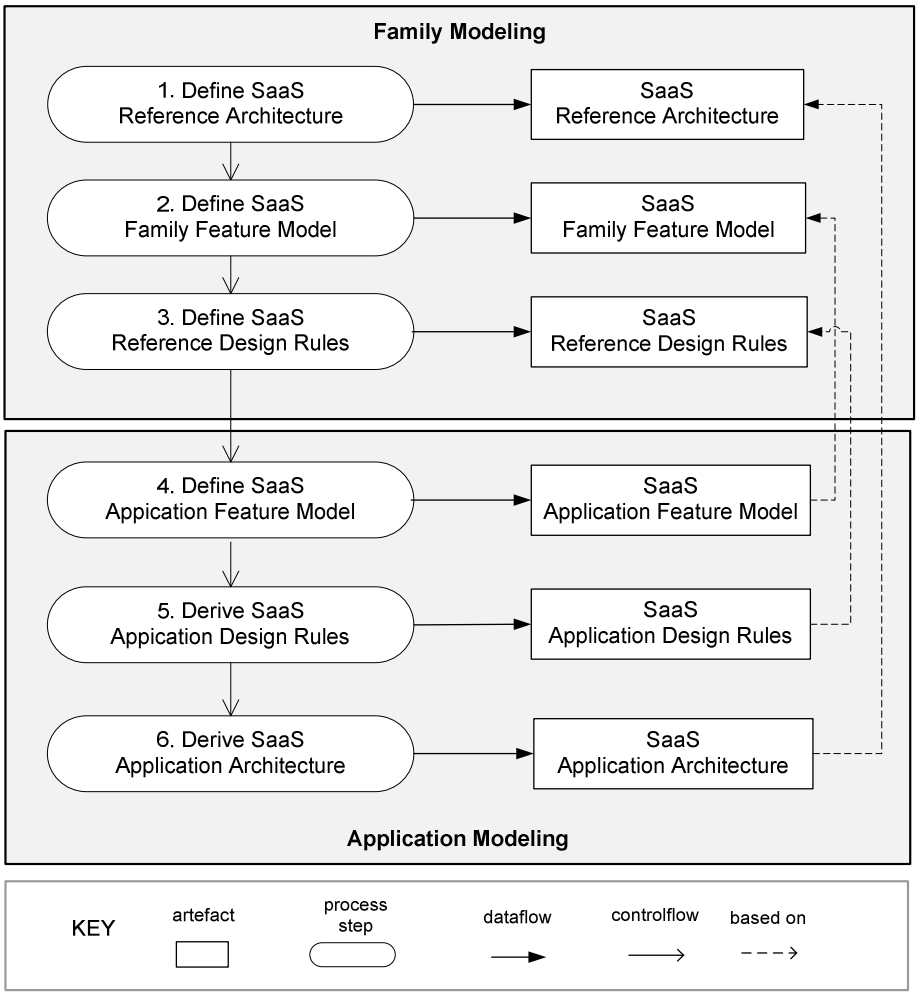


**Figure 22 – Approach for Deriving SaaS Application Architecture**

### 4.1.1 Define SaaS Reference Architecture

The first step includes the definition of the SaaS reference architecture as defined in Figure 4. The SaaS reference architecture can be further specialized if needed.

### 4.1.2 Define SaaS Family Feature Model

Feature models are often used for defining the model of products for a given application domain [64]. Feature modeling has also been extensively used in domain engineering. Hereby, a feature model is a result of a domain analysis process in which the common and variant properties of a domain are elicited and modeled. In addition, the feature model identifies the constraints on the legal combinations of features. A feature model can thus be considered as a specification of the family.

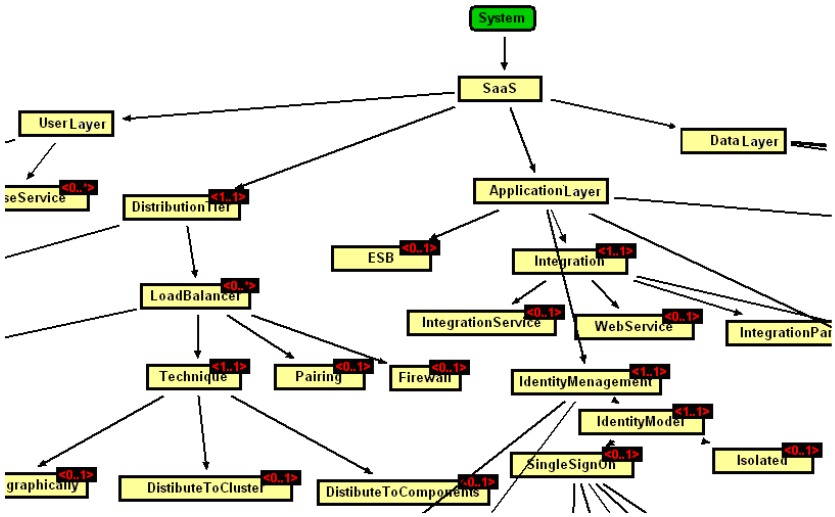


**Figure 23 – Part of the Family Feature Model for SaaS**

Part of the family feature diagram for SaaS is given in Figure 23 (due to space limitations we have not depicted the complete feature diagram). In the feature diagram SaaS is the root feature that includes four mandatory features: User Layer, Distribution Layer, Application Layer, and Data Layer. As such these features must be included in each SaaS application architecture. The SaaS family feature model is both an input for step 4 in which the application feature model is

defined based on the SaaS family feature model, and an input for step 3 in which the design rules for the selection of the features are defined.

### 4.1.3 Define SaaS Reference Design Rules

The SaaS family feature diagram represents the possible sets of feature that are required. In addition, a selection of each feature will shape the SaaS application architecture and as such represents a design decisions. As such, after defining the family feature model we define the corresponding design rules for the selection of the features. We specify the rules using the design rule definition language that we have defined as given in Figure 24.

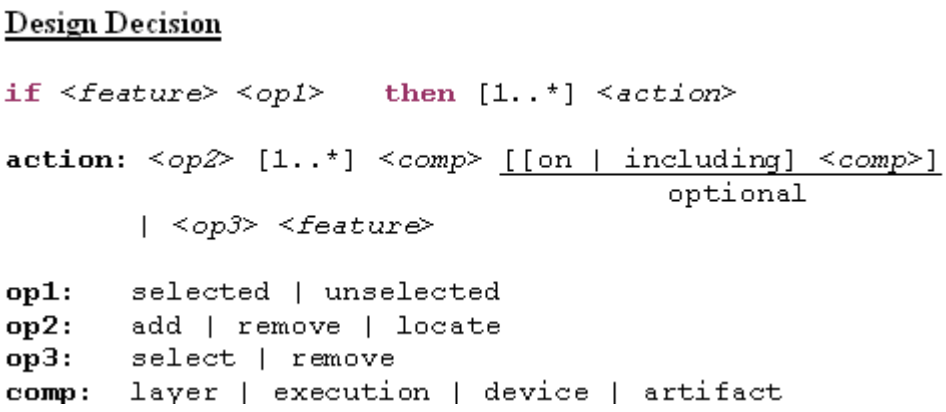


**Figure 24 – Design Rule Definition Language**

Using the design rule definition language we have specified around 30 design rules for the selection of the features in the family feature model. An example set of the rules is given in Figure 25.

### 4.1.4 Define SaaS Application Feature Model

Once the SaaS reference architecture, the family feature model, and the corresponding design rules are specified we can start the definition of the SaaS application architecture. The input for this step is basically the family feature model. Based on the requirements of the stakeholders important features are selected from the SaaS feature model.

### 4.1.5 Derive SaaS Application Design Rules

The input for this step is both the SaaS reference design rules and the application feature model. Based on the features in the application feature model we derive the corresponding application design rules from the SaaS reference design rules.

1. **if** "Client.Certified Partner" selected,
   **then** add component <<execution>> web service on <<device>> Integration Server
2. **if** "Distribution Layer.Firewall" selected,
   **then** add <<device>> firewall
3. **if** "Load Balancer.Type.Software Based" selected,
   **then** add <<device>> typical server to be used as load balancer
4. **if** "Load Balancer.Firewall" selected,
   **then** remove <<device>> Firewall
5. **if** "Application Server.Clustering" selected,
   **then** add <n> <<device>> application server
6. **if** "Multi-Tenancy.Separate DB" selected,
   **then** add <n> <<device>> Database Server
   including <<execution>> Database Engine <<artifact>> Database
7. **if** "Identity Management.Single Sign On" selected,
   **then** add <<execution>> Kerberos Authentication Server

**Figure 25 – Design Rules based on features in Family Feature Model**

# CHAPTER 5

# TOOL SUPPORT

Although the steps of the process in the previous section can be performed manually, we have developed a set of tools to assist the SaaS application design process. Figure 27 depicts the data flow and order of the steps including the tools. In the following subsections we explain the tool support activities together with a running example.

## 5.1 Feature Modeling

An important part of the process consists of feature modeling. We have used the tool XFeature [73]to define both the SaaS reference feature model and to derive the application feature model. In fact the feature model as defined in Figure 23 is a snapshot of the XFeature model. In Figure 27 the family feature modeling is defined as step 1, while the application feature modeling is defined in step 4.

Using XFeature it is possible to edit and extend the feature diagram. XFeature has a graphical editor and represents the hierarchical structure visually. The resulted family feature model is stored in xml files. The family feature model is stored in the file SaaS-FM.xfm; the application feature model is stored in Application-FM.xfm.

In Figure 26 we illustrate the feature modeling approach we show an example of the selected features based on the family feature model.

```
User Layer
------------------
User.Client
User.Certified Partner

Distribution Layer
------------------
Distribution Layer.Firewall
Load Balancer.Technique.Direct Routing
Load Balancer.Type.Hardware Based
Load Balancer.Pairing

Application and Business Service Layer
------------------
Application Layer.Enterprise Service Bus
Application Layer.Orchestration
Identity Management.LDAP
Identity Management.Single Sign On
Single Sign On.Kerberos
Single Sign On.SAML
Communication Server.Protocol.SMTP
Communication Server.Reporting
Application Server.Clustering
Integration.Common Integration
Integration.Web Service

Data Access Layer
------------------
Data Access.Cache Server
Data Access.Multitenacy.Shared

Data Storage Layer
------------------
Data Storage.Storage Area Network
```

**Figure 26 – Example Feature Model derived from Family Feature Model**

Note that in Figure 26 there are no variant features, the features for the specific business requirements have determined the selected features. As an example we can observe that for the Distribution Layer the features Firewall, Direct Routing, Hardware Based and Paring have been selected.

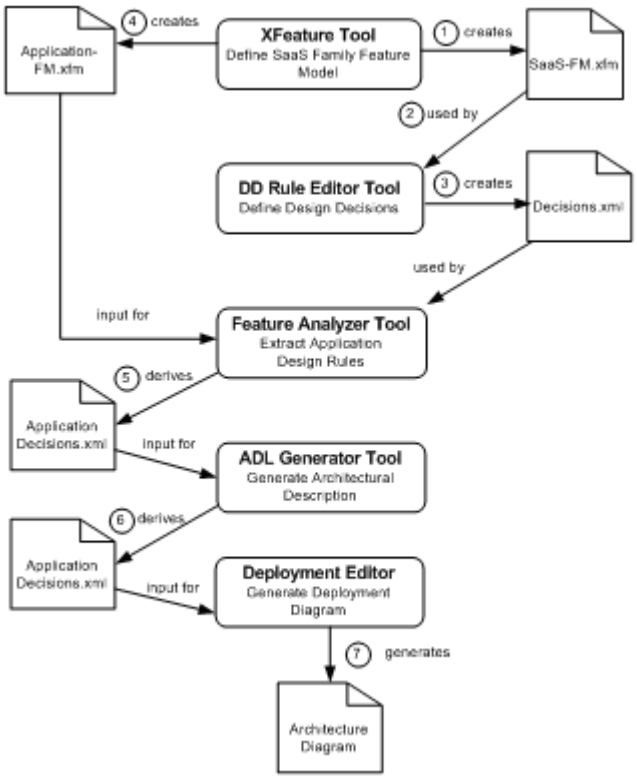


**Figure 27 – Tool Support Data Flow**

## 5.2 Design Rule Modeling

To represent the design rules we have developed a tool called Design Rule Editor which is shown in Figure 28. The tool supports the earlier defined Design Rule Definition Language and we can use it to specify the design rules for the features in the family feature model.

Design Rule Editor uses the SaaS Family Feature Model file (SaaS-FM.xfm) created in the previous step. All features from the feature model are listed and the user selects one of the features and defines the rule about that feature.

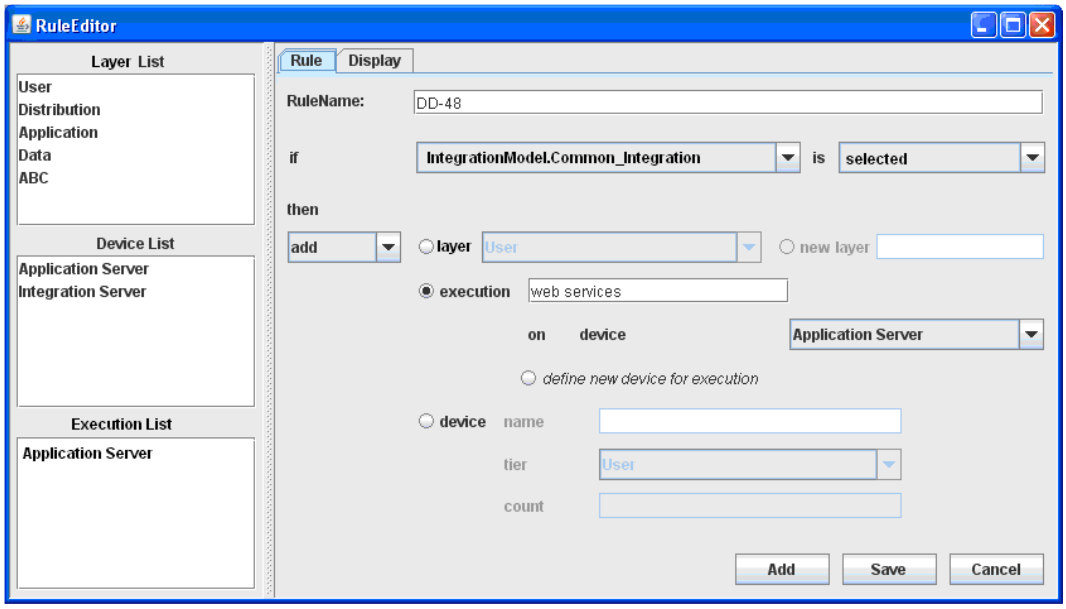


**Figure 28 – Design Decision Rule Editor**

As an example, in Figure 11 we show the definition of the rule "if Integration Model.Common_Integration is selected then add execution 'Web Service' on device Integration Server". In this case, the designer aims to provide Web services for data integration to its clients instead of implementing customer specific integration services and the rule dictates that there should be a piece of software as web services on the specified device. In the Display tab of the tool, the human-readable form of the rule is showed and the user can add note or a description of the rule. With this rule editor we have specified all the reference rules based on the family feature model which is stored in the file Decisions.xml as defined in Figure 27.

### 5.3 Associating Decisions to Features

In the previous steps we have generated an application feature model (stored in Application-FM.xfm) and we have defined the design decisions rules (stored in Decisions.xml). In this step we correlate the design decisions to features using again the XFeature Tool. For this we define a new attribute in the feature model for design decisions. The user can enter the identifier of the design decision rule as an attribute for the feature from properties panel. A snapshot of the XFeature for this step is shown in Figure 29.

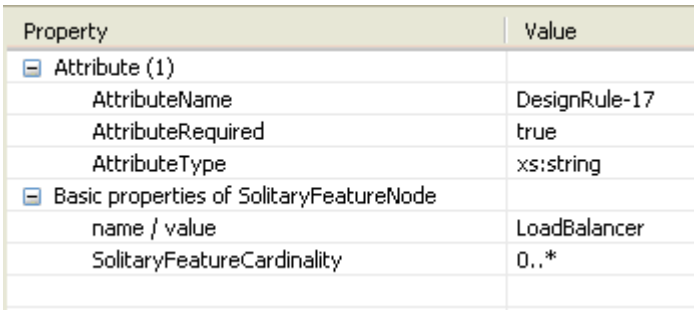


**Figure 29 – Mapping Design Rules to Features from Properties Panel**

For the example application feature model in Figure 26 the design rules have been derived by checking the reference design rules and matching it with the selected features. We show, as an example, the set of the derived rules for the Application and Business Layer and the Data Access and Storage Layer features:

```
if "Application Layer.Enterprise Service Bus" selected then add <device>
ESB including <execution> ESB Services
if "Application Layer.Orchestration" selected then add <execution>
Orchestration Service on <device> ESB
if "Identity Management.LDAP" selected then add <device> LDAP server
if "Identity Management.Single Sign On" selected then add <device>
"Identity Management Server" including <execution> Identity Management
System
if "Single Sign On.Kerberos" selected then add <device> Kerberos Server
if "Application Server.Clustering" selected then add <n> <device>
Application Server
if "Integration.Web Service" selected then add <device> Integration Server
if "Integration.Common Integration" selected then add <execution> Web
service on <device> Integration Server
if "Data Access.Cache Server" selected then add <n> <device> Cache Server
if "Data Access.Multitenacy.Shared" selected then add <1> <execution> DBMS
if "Data Storage.Storage Area Network" selected then <n> <device> Storage
Device
```

**Figure 30 – Derived Rules based on the selected features in Figure 26**

After the correlation of the design rules and features, the next step is creating an instance of the family, which is called application model.

### 5.4 ADL Generation

In this study, we aimed to provide guidance for reasoning about alternative SaaS architectures. So far, we were able to define an application model from the family and we need to represent the corresponding architecture of the application model. The design decision rules, we mentioned before, are useful for exposing the architecture. Since application model derives from the family model, it also inherits the attributes. Within the application model features, there are references to design rules as attributes. Here, we introduce another tool, Feature Analyzer which takes as input both the application model file (Application-FM.xfm) and design decision rules file (Decisions.xml). The tool automatically extracts the attributes of the features, finds references to design rules and links it to those rules. As a result, all features of the application model are represented graphically

as a tree-like hierarchical structure and the corresponding design decision rules are displayed.

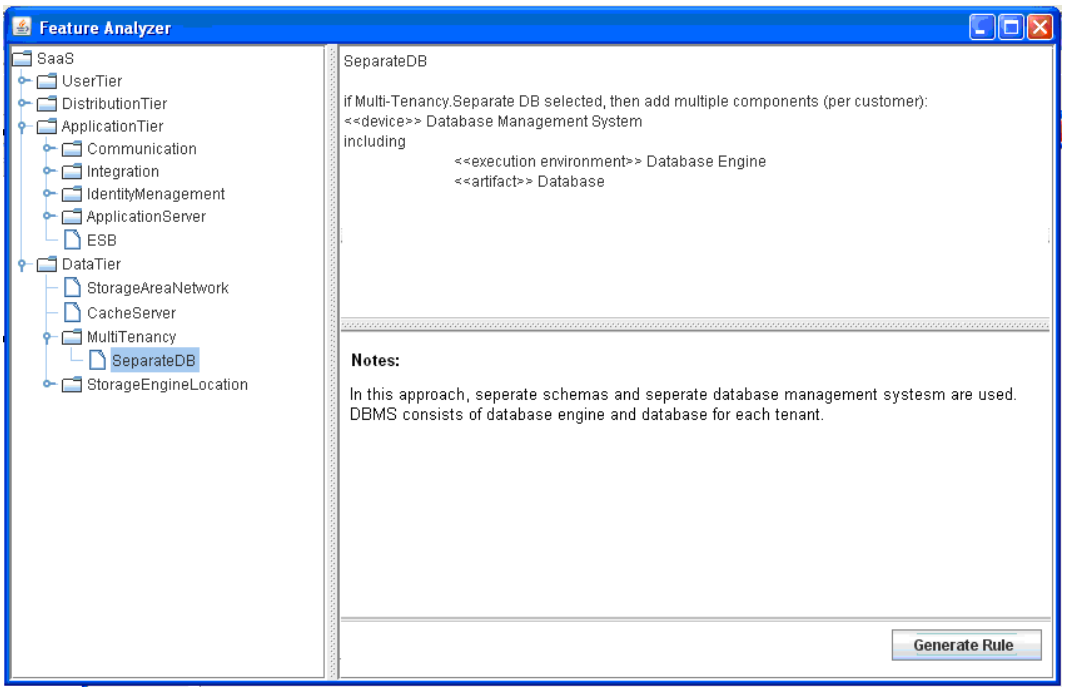


**Figure 31 – Design Feature Analyzer Tool**

As shown in Figure 31, on the left side of the panel, the features are displayed for a specific alternative application model. In case of selecting a feature, the corresponding design rule is displayed on the right side. Remember that, Design Rule Editor allows adding notes for the features and the notes are also displayed on the panel.

```
- <Device name="AppServer" id="0" tier="Application">
    <Execution id="10">ApplicationServer</Execution>
    <Execution id="11">MetricServer</Execution>
  </Device>
- <Device name="IntegrationServer" id="1" tier="Application">
    <Execution id="12">WebServer</Execution>
  </Device>
  <Device name="CommunicationServer" id="2" tier="Application" />
  <Connection srcID="0" destID="1" />
  <Connection srcID="2" destID="0" />
```

**Figure 32 –An Example of ADL that we used**

The next step is transforming these design rules to architecture. For this, we have developed a simple architecture description language (ADL) [71]. This language has only basic types for describing the architecture: device, execution and connection. This ADL is used internally, that is to say, the user does not write a description manually. We have developed another tool, ADL Generator, which takes application specific design rules and converts it to architectural description in xml format. A part of the architectural description is shown in Figure 32 which is generated by the ADL Generator Tool.

### 5.5 Generating Deployment Diagram for SaaS Architecture

The final step is showing graphical view of the architecture. Deployment diagram is a static view of the hardware, the software running on that hardware and the relationship between them. We have chosen the deployment view of the architecture to display, because deployment diagram is also very useful for system engineering. It can be used for analyzing quality attributes such as scalability, performance, maintainability, portability, and so on [64]. We have developed an eclipse plug-in [67], an editor, which is capable of both drawing deployment diagram automatically from ADL and enabling user for editing the generated diagram.

We used Model Driven Architecture (MDA) and Eclipse Graphical Modeling Framework (GMF) [66] for developing deployment diagram editor. MDA provides high level abstraction, platform independent modeling approach and uses a Domain Specific Language. GMF helps to define domain models and represent graphically based on MDA.

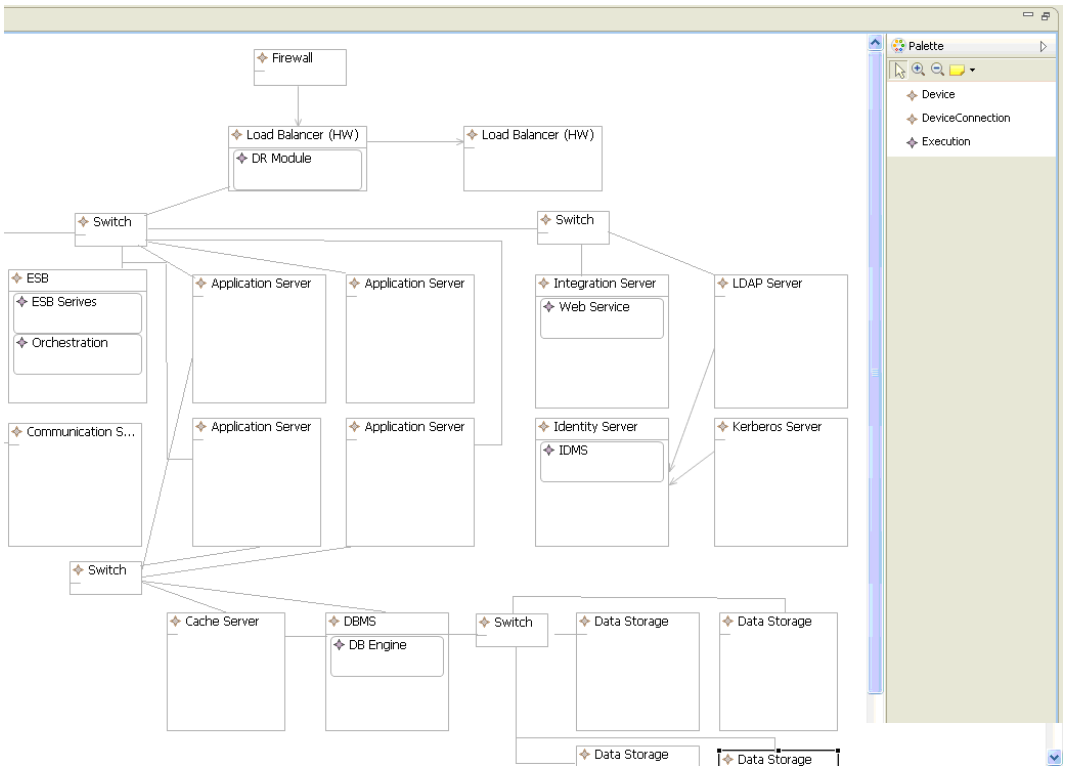


**Figure 33 – Design Deployment Diagram Editor**

In our ADL, we have basic elements to define architecture. To develop the deployment diagram, we also need DSL elements that correspond to ADL elements. Thus, device, execution and connector model and meta-model files are defined in GMF. By using the model and meta-model files, GMF generates the tool code.

The graphical editor generates the deployment diagram automatically from the architectural description which is generated in the previous step. First, the editor parses the ADL components than determines the layout of the components and arranges the position of the components.

After the deployment diagram is generated automatically, the user can modify the diagram arbitrary. Figure 16 illustrates the visual representation of the architecture by the deployment diagram editor for the example application feature model of Figure 26 and the derived application.

# CHAPTER 6

# CONCLUSION AND FUTURE WORK

Based on the literature we can define a common SaaS architecture that includes the concepts and relations among the concepts to derive SaaS application architectures. Selecting the appropriate SaaS architecture is important to meet the specific business requirements. Unfortunately the current state-of-the art does not provide an explicit approach for guiding the selection of the application architecture.

We think that there are two important contributions in the study. First of all we provide a systematic approach for modeling SaaS reference architecture and deriving different SaaS application architecture based on the selections of features from family feature model. The mechanism for distinguishing the modeling between family modeling and application modeling appeared to be very useful. In the family modeling part we actually applied a domain engineering process and defined the reference architecture, the family feature model and the reference design rules. The reference architecture actually defines the space of application architectures. The family feature model defines the possible features for SaaS applications, and it appeared that we can relate these to specific architectural decisions. Based on the derived architectural decisions we could derive the specific application architecture.

A second important contribution is the toolset that we have developed for supporting the process. With the toolset we actually store the complete derivation of the application architecture from the feature models. The application features, the derived design rules and the eventual application architecture are linked to each other and as such the design decisions and the requirements feature selection for the application architecture can be easily traced. By defining multiple application architectures based on different application feature models we can even compare multiple alternatives and based on this select the most feasible alternative.

There are also many possible extensions possible for this work. Our future work will in the first place focus on applying the process within an industrial context. We will also extend our toolset and provide a full integration of the tools within an Eclipse development environment. Finally, we will also focus on nonfunctional requirements in selecting application architectures [64]. In this study we have mainly focused on functional feature set as defined in the family feature model. We think that we can equally focus on quality feature models to derive the application architecture.

# REFERENCES


[1] Mark Turner, David Budgen, Pearl Brereton, "Turning Software into a Service," Computer, vol. 36, no. 10, pp. 38-44, Oct. 2003, doi:10.1109/MC.2003.1236470

[2] N. "Gold, A. Mohan, C. Knight, M. Munro; "Understanding service-oriented software", IEEE Software archive, Volume 21, Issue 2 (March 2004), Pages: 71 - 77, Year of Publication: 2004, ISSN:0740-7459

[3] D. Budgen, P. Brereton, M. Turner; "Codifying a service architectural style", Proceedings of the 28th Annual International Computer Software and Applications Conference - Volume 01, Pages: 16 - 22, Year of Publication: 2004, ISBN ~ ISSN:0730-3157 , 0-7695-2209-2-1

[4] J. Sathyan, K. Shenoy; "Realizing unified service experience with SaaS on SOA", Proceedings of the Third International Conference on Communication System Software and Middleware (COMSWARE'08)

[5] Wei Sun, Xin Zhang, Chang Jie Guo, Pei Sun, Hui Su; "Software as a Service: Configuration and Customization Perspectives", Proceedings of the 2008 IEEE Congress on Services Part II, Pages: 18-25 , Year of Publication: 2008, ISBN:978-0-7695-3313-1

[6] E. Knorr; "Software as a Service: The Next Big Thing", http://www.infoworld.com/d/applications/software-service-next-big-thing-319, Last accessed on June 2010

[7] Vidyanand Choudhary; "Software as a Service: Implications for Investment in Software Development", HICSS, Proceedings of the 40th Annual Hawaii International Conference on System Sciences, Page: 209a, Year of Publication: 2007, ISBN ~ ISSN:1530-1605 , 0-7695-2755-8

[8] SIIA; "Software as a Service: Strategic Backgrounder", http://www.siia.net/estore/pubs/SSB-01.pdf, Last accessed on June 2010

[9] K.Bennett, P. Layzell, D. Budgen, P. Brereton, L. Macaulay, M. Munro; "Service-Based Software: The Future for Flexible Software", APSEC, Proceedings of the Seventh Asia-Pacific Software Engineering Conference, Page: 214, Year of Publication: 2000, ISBN:0-7695-0915-0

[10] Progress, "Saas Architecture White Paper", http://communities.progress.com/pcom/community/psdn/openedge/architecture, Last accessed on June 2010

[11] "Architecture Strategies for Catching the Long Tail", http://msdn.microsoft.com/en-us/library/aa479069.aspx, Last accessed on June 2010

[12] "What is SaaS?", http://www.parallels.com/eu/spp/understandingsaas/, Last accessed on June 2010

[13] Chang jie Guo, etc; “A Framework for Native Multi-Tenancy Application Development and Management”,CEC/EEE 2007

[14] "Software as a Service", http://en.wikipedia.org/wiki/Software_as_a_Service

[15] V. Singla; "The Overlapping Worlds of SaaS and SOA", http://cloudcomputing.sys-con.com/node/1047073/, Last accessed on June 2010

[16] SaaS-Attack.com; “SaaS – Top 10 ISV Concerns”

[17] What is SaaS?, http://www.tech-faq.com/saas.shtml, Last accessed on June 2010

[18] "Software as a Service (SaaS): An Enterprise Perspective", http://msdn.microsoft.com/en-us/library/aa905332.aspx, Last accessed on June 2010

[19] SaaS-Attack.com, “Top 10 Design Issues”, Last accessed on June 2010

[20] Serice-now.com, “SaaS vs. ASP”, http://www.slideshare.net/rglauser/servicenowcom-saas-vs-asp-vs-traditional-software-presentation, Last accessed on June 2010

[21] “SOA and SaaS ... What's the Difference?”, http://software.intel.com/en-us/blogs/2008/08/18/soa-and-saas-whats-the-difference/, Last accessed on June 2010

[22] Phillip 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, ISSN:1520-9202

[23] "Software as a Service vs. Service Oriented Architecture vs. Cloud Computing", http://www.galorath.com/wp/software-as-a-service-vs-service-oriented-architecture-vs-cloud-computing.php/comment-page-1, Last accessed on June 2010

[24] "Are SaaS & Cloud Computing Interchangeable Terms? ", http://www.daniweb.com/blogs/entry3993.html#, Last accessed on June 2010

[25] "Cloud computing", http://en.wikipedia.org/wiki/Cloud_computing, Last accessed on June 2010

[26] D. Fishteyn; "Deploying Software as a Service (SaaS)", http://www.saas.com, Last accessed on June 2010

[27] Progress, "Architecting for Software as a Service", http://communities.progress.com/pcom/community/psdn/openedge/architecture, Last accessed on June 2010

[28] "ISVs are from Mars, and Hosters are from Venus", http://msdn.microsoft.com/en-us/library/bb891759.aspx, Last accessed on June 2010

[29] "Architecture Strategies for Catching the Long Tail", http://msdn.microsoft.com/en-us/library/aa479069.aspx, Last accessed on June 2010

[30] Vivek Bhatnagar, "Understanding SaaS Architecture", http://blogs.msdn.com/b/gianpaolo/archive/2006/10/03/understanding-saas-architecture-powerpoint-presentation.aspx, Last accessed on June 2010

[31] Progress.com, "Saas Scalability White Paper", http://communities.progress.com/pcom/community/psdn/openedge/architecture, Last accessed on June 2010

[32] André B. Bondi; "Characteristics of scalability and their impact on performance", Workshop on Software and Performance, Proceedings of the 2nd international workshop on Software and performance, Ottawa, Ontario, Canada, Pages: 195 - 203, Year of Publication: 2000, ISBN:1-58113-195-X

[33] Scalability White Paper – I, http://www.saas.com, Last accessed on June 2010

[34] "Scalability", http://en.wikipedia.org/wiki/Scalability, Last accessed on June 2010

[35] "What is Scalability?", http://searchdatacenter.techtarget.com/sDefinition/0,,sid80_gci212940,00.html, Last accessed on June 2010

[36] Jeff Offutt, "Quality Attributes of Web Software Applications", IEEE Software, Volume 19 , Issue 2 (March 2002), Pages: 25 - 32, Year of Publication: 2002, ISSN:0740-7459"

[37] "Scalability", http://fox.wikis.com/wc.dll?Wiki~Scalability, Last accessed on June 2010

[38] John Udell, "IT Myth 6: IT doesn't scale"; http://www.infoworld.com/d/developer-world/it-myth-6-it-doesnt-scale-007, Last accessed on June 2010

[39] "Scalability Requirements", http://fox.wikis.com/wc.dll?Wiki~ScalabilityRequirements, Last accessed on June 2010

[40] Richard Winter; "Lexicology of Scale", http://www.wintercorp.com/rwintercolumns/ie_9902.html, Last accessed on June 2010

[41] "Scalability Principles", http://www.infoq.com/articles/scalability-principles, Last accessed on June 2010

[42] "Scalability Guidelines", http://www.hfadeel.com/Blog/?p=120, Last accessed on June 2010

[43] Gunnar Brataas, Peter Hughes; "Exploring Architectural Scalability", Last accessed on June 2010

[44] Prasad Jogalekar, Murray Woodside; "Evaluating the Scalability of Distributed Systems", IEEE Transactions on Parallel and Distributed Systems, Volume 11, Issue 6 (June 2000), Pages: 589 - 603, Year of Publication: 2000, ISSN:1045-9219

[45] BEAWebLogic, "Capacity Planning and Performance Tuning", http://download.oracle.com/docs/cd/E12530_01/oam.1014/b32419.pdf, Last accessed on June 2010

[46] "Fault-tolerant system", http://en.wikipedia.org/wiki/Fault-tolerant_system, Last accessed on June 2010

[47] "Scalability Overview", http://msdn2.microsoft.com/en-us/library/aa292203(VS.71).aspx, Last accessed on June 2010

[48] Theo Schlossnagle, "Scalable Internet Architectures", Publisher: Sams (July 31, 2006); ISBN-10: 067232699X, ISBN-13: 978-0672326998

[49] "What is Scalability?", http://www.adobe.com/livedocs/coldfusion/5.0/Advanced_ColdFusion_Administration/overview2.htm, Last accessed on June 2010

[50] Tom Atwood, "Vertical and Horizontal Computing Architectures - Trends and Attributes", http://sun.systemnews.com/articles/65/5/feature/10540, Last accessed on June 2010

[51] Gianpaolo Carraro; "SaaS Guidance", http://blogs.msdn.com/gianpaolo/archive/2006/05/30/610611.aspx, Last accessed on June 2010

[52] Gianpaolo Carraro; "SaaS Architectures", http://blogs.msdn.com/gianpaolo/archive/2006/10/01/SaaS-Architectures.aspx, Last accessed on June 2010

[53] "An Overview of Software as a Service in Retail", http://msdn.microsoft.com/en-us/library/bb507203.aspx, Last accessed on June 2010

[54] "Software as a Service (SaaS): An Enterprise Perspective", http://msdn.microsoft.com/en-us/library/aa905332.aspx, Last accessed on June 2010

[55] Kanwardeep Singh Ahluwalia, "Scalability Design Patterns", www.hillside.net/plop/2007/papers/PLoP2007_Ahluwalia.pdf, Last accessed on June 2010

[56] Progress, “Application Tenancy White Paper”, http://communities.progress.com/pcom/community/psdn/openedge/architecture, Last accessed on June 2010

[57] “Enterprise application integration”, http://en.wikipedia.org/wiki/Enterprise_application_integration, Last accessed on June 2010

[58] Anthony T. Velte, Toby J. Velte, Robert Elsenpeter; Cloud Computing: A Practical Approach, Publisher: McGraw-Hill Osborne Media; 1 edition (September 22, 2009), ISBN-10: 0071626948, ISBN-13: 978-007162694

[59] Sun, "Cloud Computing Primer", http://www.scribd.com/doc/18824914/Cloud-Computing-Primer, Last accessed on June 2010

[60] Jon Brodkin, “Gartner: Seven cloud-computing security risks”, InfoWord, http://www.infoworld.com/article/08/07/02/Gartner_Seven_cloudcomputing_security_risks_1.html, , Last accessed on June 2010

[61] GTSI, "Cloud Computing: Building a Framework for Successful Transition", http:www.gtsi.com/cms/documents/White-Papers/Cloud-Computing.pdf, Last accessed on June 2010

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

[63] F. Bachmann, L. Bass, M. Klein, Deriving Architectural Tactics: A Step Toward Methodical Architectural Design, CMU/SEI-2003-TR-004, ADA413644, Pittsburgh, PA, March 2003.

[64] S. Balsamo, A. Di Marco, P. Inverardi, M. Simeoni, Model-Based Performance Prediction in Software Development: A Survey, IEEE Transactions on Software Engineering, vol. 30, no. 5, pp. 295-310, May 2004.

[65] K. Czarnecki and U. Eisenecker. Generative Programming: Methods, Tools, and Applications, Addison Wesley, 2000.

[66] Eclipse Modeling Framework Web Site, http://www.eclipse.org/emf/, Last accessed on June 2010

[67] Eclipse official web site, http://www.eclipse.org, Last accessed on June 2010

[68] M. Godse, S. Mulik, An Approach for Selecting Software-as-a-Service (SaaS) Product, pp.155-158, 2009 IEEE International Conference on Cloud Computing, 2009.

[69] G. Goth. Software-as-a-Service: The Spark That Will Change Software Engineering? IEEE Distributed Systems Online, vol. 9, no. 7, 2008.

[70] H. Liao. Design of SaaS-Based Software Architecture, niss, pp.277-281, 2009 International Conference on New Trends in Information and Service Science, 2009.

[71] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. IEEE Trans. Software Eng., 26(1):70–93, 2000.

[72] M. Turner , D. Budgen, P. Brereton, Turning Software into a Service, Computer, v.36 n.10, p.38-44, October 2003.

[73] XFeature official web site, http://www.pnp-software.com/XFeature, , Last accessed on June 2010

[74] Simple Authentication and Security Layer, http://asg.web.cmu.edu/sasl/, Last accessed on June 2010

[75] XML Gateway - Load balancing Techniques, http://xml-gateway.blogspot.com/2009/07/xml-gateway-load-balancing-techniques.html, Last accessed on June 2010

[76] jetNEXUS Application Load Balancer, http://www1.jetnexus.com/web-load-balancer.html, Last accessed on June 2010

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

[78] MSDN, "Server Clustering", http://msdn.microsoft.com/en-us/library/ff649250.aspx, Last accessed on June 2010

# APPENDIX A

## SAMPLE USAGE OF THE APPROACH AND TOOLS

The process for modeling application architecture is depicted at Figure 22. The steps are summarized as below:

1. Define SaaS Reference Architecture
2. Define SaaS Family Feature Model
3. Define SaaS Reference Design Rules
4. Define SaaS Application Feature Model
5. Derive SaaS Application Design Rules
6. Derive SaaS Application Architecture

According to this process, we can divide the steps in two parts. The first three steps are common for each application architecture. In Chapter 3, we defined the SaaS Reference Architecture and SaaS Family Feature Model. For the third step, here is the complete list of SaaS Reference Design Rules:

- *DD-1 = if [Distribution Layer.Firewall] is selected, then add [n] <device> "Firewall" on <layer> "Distribution"*
- *DD-2 = if [Load Balancer.Type.Hardware Based] is selected, then add [n] <device> "Hardware LB" on <layer> "Distribution"*
- *DD-3 = if [Load Balancer.Type.Software Based] is selected, then add <device> "Software LB" on <layer> "Distribution"*
- *DD-4 = if [Presentation Layer.Web Server] is selected, then add [n] <device> "Web Server" on <layer> "Presentation"*

- *DD-5 = if [Presentation Layer.Web Proxy Server] is selected, then add [n] <device> "Web Proxy Server" on <layer> "Presentation"*
- *DD-6 = if [Application Layer.Integration] is selected, then add [n] <device> "Integration Server" on <layer> "Application"*
- *DD-7 = if [Integration.Service.Integration Service] is selected, then add <execution> "Integration Service" on <device> "Integration Server"*
- *DD-8 = if [Integration.Service.Web Service] is selected, then add <execution> "Web Service" on <device> "Integration Server"*
- *DD-9 = if [Application Layer.ESB] is selected, then add [n] <device> "ESB Server" including <execution> "ESB" on <layer> "Application"*
- *DD-10 = if [Application Layer.BPEL Engine] is selected, then add <execution> "BPEL Engine"*
- *DD-11 = if [Application Layer.Business Rule Engine] is selected, then add <execution> "Business Rule Engine"*
- *DD-12 = if [Application Layer.Metadata Management] is selected, then add <execution> "Metadata Management Service"*
- *DD-13 = if [Application Layer.Application Server] is selected, then add [1] <device> "Application Server" on <layer> "Application"*
- *DD-14 = if [Clustering.Asymetric] is selected, then add [n] <device> "Application Server" on <layer> "Application" and add <device> "Standby Server" on <layer> "Application"*
- *DD-15 = if [Clustering.Symetric] is selected, then add [n] <device> "Application Server" on <layer> "Application"*
- *DD-16 = if [Application Layer.Identity Management] is selected, then add [n] <device> "IdM Server" on <layer> "Application"*
- *DD-17 = if [Token-based.Kerberos] is selected, then add <execution> "Kerberos Authentication Server" on <device> "IdM Server"*

- *DD-18 = if [Method.Kerberos-based] is selected, then add <execution> "Kerberos Authentication Server" on <device> "IdM Server"*
- *DD-19 = if [Application Layer.Communication] is selected, then add [n] <device> "Communication Server" on <layer> "Application"*
- *DD-20 = if [Communication.MTA] is selected, then add <execution> "MTA" on <device> "Communication Server"*
- *DD-21 = if [Separate Databases] is selected, then add [n] <device> "DBMS" including <execution> "Database Engine" on <layer> "Data"*
- *DD-22 = if [Shared DB Separate Schema] is selected, then add <device> "DBMS" including <execution> "Database Engine"<device> on <layer> "Data"*
- *DD-23 = if [Shared DB Shared Schema] is selected, then add <device> "DBMS" including <execution> "Database Engine" on <layer> "Data"*
- *DD-24 = if [Metadata Repository] is selected, then add [n] <device> "Metadata DB" on <layer> "Data"*
- *DD-25 = if [Directory Server] is selected, then add [n] <device> "Directory Server" on <layer> "Data"*
- *DD-26 = if [Application Database] is selected, then add <device> "DB Server" on <layer> "Data"*
- *DD-27 = if [Application Database.Clustering] is selected, then add [n] <device> "DB Server" on <layer> "Data"*
- *DD-28 = if [SAN] is selected, then add <device> "SAN" on <layer> "Data"*
- *DD-29 = if [Caching] is selected, then add <device> "Cache Server" on <layer> "Data"*

We used XFeature tool for creating the family feature model. This tool provides additional capabilities, such as adding attributes to features. Since we aim to generate a deployment diagram, we defined some new attributes for specifying number of devices.

The fourth step is deriving an application feature model from the family feature model. While defining application feature model, the constraints are enabled. For the sample scenario, the features listed below are selected:

- User Layer
    - Web Browser
- Distribution Layer
    - Firewall
    - Load Balancer
        - Type
            - Hardware Based
        - Strategy
            - Passive
                - Round Robin
- Presentation Layer
    - Web Server
- Application Layer
    - ESB
    - BPEL Engine
    - Business Rule Engine
    - Metadata Management
    - Integration
        - Method
            - Common
        - Service
            - Web Service

        - Application Server
            - Clustering
                - Symmetric
        - Identity Management
            - Identity Model
                - Federated
                    - Method
                        - Kerberos-based
            - Directory Management
                - Protocol
                    - LDAP
        - Communication
            - MTA
                - Protocol
                    - SMTP
- Data Access Layer
    - Multi-tenancy
        - Shared DB Separate Schema
- Data Storage Layer
    - Metadata Repository
    - Directory Server
    - Application Database
        - Clustering

According to these selections, the following design rules are brought in the fifth step:

- *DD-1 = if [Distribution Layer.Firewall] is selected, then add [n] <device> "Firewall" on <layer> "Distribution"*

- *DD-2 = if [Load Balancer.Type.Hardware Based] is selected, then add [n] <device> "Hardware LB" on <layer> "Distribution"*
- *DD-4 = if [Presentation Layer.Web Server] is selected, then add [n] <device> "Web Server" on <layer> "Presentation"*
- *DD-6 = if [Application Layer.Integration] is selected, then add [n] <device> "Integration Server" on <layer> "Application"*
- *DD-8 = if [Integration.Service.Web Service] is selected, then add <execution> "Web Service" on <device> "Integration Server"*
- *DD-9 = if [Application Layer.ESB] is selected, then add [n] <device> "ESB Server" including <execution> "ESB" on <layer> "Application"*
- *DD-10 = if [Application Layer.BPEL Engine] is selected, then add <execution> "BPEL Engine"*
- *DD-11 = if [Application Layer.Business Rule Engine] is selected, then add <execution> "Business Rule Engine"*
- *DD-12 = if [Application Layer.Metadata Management] is selected, then add <execution> "Metadata Management Service"*
- *DD-13 = if [Application Layer.Application Server] is selected, then add [1] <device> "Application Server" on <layer> "Application"*
- *DD-15 = if [Clustering.Symetric] is selected, then add [n] <device> "Application Server" on <layer> "Application"*
- *DD-16 = if [Application Layer.Identity Management] is selected, then add [n] <device> "IdM Server" on <layer> "Application"*
- *DD-18 = if [Method.Kerberos-based] is selected, then add <execution> "Kerberos Authentication Server" on <device> "IdM Server"*
- *DD-19 = if [Application Layer.Communication] is selected, then add [n] <device> "Communication Server" on <layer> "Application"*
- *DD-20 = if [Communication.MTA] is selected, then add <execution> "MTA" on <device> "Communication Server"*

- *DD-22 = if [Shared DB Separate Schema] is selected, then add <device> "DBMS" including <execution> "Database Engine"<device> on <layer> "Data"*
- *DD-24 = if [Metadata Repository] is selected, then add [n] <device> "Metadata DB" on <layer> "Data"*
- *DD-25 = if [Directory Server] is selected, then add [n] <device> "Directory Server" on <layer> "Data"*
- *DD-26 = if [Application Database] is selected, then add <device> "DB Server" on <layer> "Data"*
- *DD-27 = if [Application Database.Clustering] is selected, then add [n] <device> "DB Server" on <layer> "Data"*
- *DD-29 = if [Caching] is selected, then add <device> "Cache Server" on <layer> "Data"*

As seen, some rules are incomplete. For example we have to specify where the BPEL Engine will run on and how much servers will be used for clustering. By the Design Rule Editor tool, we can specify them, rename features, and change the deployment. After editing the rules, final design decisions are as below:

- *if [Distribution Layer.Firewall] is selected, then add [1] <device> "Firewall" on <layer> "Distribution"*
- *if [Load Balancer.Type.Hardware Based] is selected, then add [1] <device> "Load Balancer (HW)" on <layer> "Distribution"*
- *if [Presentation Layer.Web Server] is selected, then add [1] <device> "Web Server" on <layer> "Presentation"*
- *if [Application Layer.Integration] is selected, then add [1] <device> "Integration Server" on <layer> "Application"*
- *if [Integration.Service.Web Service] is selected, then add <execution> "Web Service" on <device> "Integration Server"*

- *if [Application Layer.ESB] is selected, then add [1] <device> "ESB" including <execution> "ESB Services" on <layer> "Application"*
- *if [Application Layer.BPEL Engine] is selected, then add <execution> "BPEL Engine" on <device> "Server" on <layer> "Application"*
- *if [Application Layer.Business Rule Engine] is selected, then add <execution> "Business Rule Engine" on <device> "Server" on <layer> "Application"*
- *if [Application Layer.Metadata Management] is selected, then add <execution> "Metadata Services" on <device> "Server" on <layer> "Application"*
- *if [Application Layer.Application Server] is selected, then add [1] <device> "Application Server" on <layer> "Application"*
- *if [Clustering.Symetric] is selected, then add [3] <device> "Application Server" on <layer> "Application"*
- *if [Application Layer.Identity Management] is selected, then add [1] <device> "IdM Server" on <layer> "Application"*
- *if [Method.Kerberos-based] is selected, then add <execution> "Kerberos Authentication Server" on <device> "IdM Server"*
- *if [Application Layer.Communication] is selected, then add [1] <device> "Communication Server" on <layer> "Application"*
- *if [Communication.MTA] is selected, then add <execution> "MTA" on <device> "Communication Server"*
- *if [Shared DB Separate Schema] is selected, then add <device> "DBMS" including <execution> "Database Engine"<device> on <layer> "Data"*
- *if [Metadata Repository] is selected, then add [1] <device> "Metadata Repo" on <layer> "Data"*
- *if [Directory Server] is selected, then add [1] <device> "Directory Server" on <layer> "Data"*

- *if [Application Database] is selected, then add <device> "DB Server" on <layer> "Data"*
- *if [Application Database.Clustering] is selected, then add [4] <device> "DB Server" on <layer> "Data"*
- *if [Caching] is selected, then add <device> "Cache Server" on <layer> "Data"*

For the final step, first we generate the ADL for these design rules. As a result, the following xml based architecture description language is formed.

```
<?xml version="1.0" encoding="UTF-8"?>
<Deployment>
    <Device name="Firewall" id="0" layer="Distribution"></Device>
    <Device name="Load Balancer (HW)" id="1" layer="Distribution"></Device>
    <Device name="Web Server" id="2" layer="Presentation"></Device>
    <Device name="Web Server" id="3" layer="Presentation"></Device>
    <Device name="Web Server" id="4" layer="Presentation"></Device>
    <Device name="ESB Server" id="5" layer="Application">
        <Execution id="0">ESB Services</Execution>
    </Device>
    <Device name="Application Server" id="6" layer="Application"></Device>
    <Device name="Application Server" id="7" layer="Application"></Device>
    <Device name="Application Server" id="8" layer="Application"></Device>
    <Device name="Application Server" id="9" layer="Application"></Device>
    <Device name="Integration Server" id="10" layer="Application">
        <Execution id="1">Web Service</Execution>
    </Device>
    <Device name="Communication Server" id="11" layer="Application">
        <Execution id="2">MTA</Execution>
    </Device>
    <Device name="IdM Server" id="12" layer="Application">
```

```
        <Execution id="3">IDMS</Execution>
</Device>
<Device name="Server" id="13" layer="Application">
        <Execution id="4">BPEL Engine</Execution>
        <Execution id="5">Business Rules Engine</Execution>
        <Execution id="6">Metadata Services</Execution>
</Device>
<Device name="DBMS" id="14" layer="Data">
        <Execution id="7">DB Engine</Execution>
</Device>
<Device name="Cache Server" id="15" layer="Data"></Device>
<Device name="LDAP Server" id="16" layer="Data"></Device>
<Device name="Metadata Repo" id="17" layer="Data"></Device>
<Device name="DB Server" id="18" layer="Data"></Device>
<Device name="DB Server" id="19" layer="Data"></Device>
<Device name="DB Server" id="20" layer="Data"></Device>
<Device name="DB Server" id="21" layer="Data"></Device>
<Device name="Switch" id="22" layer="Application"></Device>
<Device name="Switch" id="23" layer="Data"></Device>
<Connection srcID="0" destID="1"/>
<Connection srcID="1" destID="2"/>
<Connection srcID="1" destID="3"/>
<Connection srcID="1" destID="4"/>
<Connection srcID="22" destID="2"/>
<Connection srcID="22" destID="3"/>
<Connection srcID="22" destID="4"/>
<Connection srcID="22" destID="5"/>
<Connection srcID="22" destID="6"/>
<Connection srcID="22" destID="7"/>
<Connection srcID="22" destID="9"/>
<Connection srcID="22" destID="10"/>
<Connection srcID="22" destID="11"/>
```

```
        <Connection srcID="22" destID="12"/>
        <Connection srcID="22" destID="13"/>
        <Connection srcID="23" destID="6"/>
        <Connection srcID="23" destID="7"/>
        <Connection srcID="23" destID="8"/>
        <Connection srcID="23" destID="9"/>
        <Connection srcID="23" destID="14"/>
        <Connection srcID="23" destID="13"/>
        <Connection srcID="23" destID="15"/>
        <Connection srcID="23" destID="16"/>
        <Connection srcID="23" destID="17"/>
        <Connection srcID="23" destID="18"/>
        <Connection srcID="23" destID="19"/>
        <Connection srcID="23" destID="20"/>
        <Connection srcID="23" destID="21"/>
</Deployment>
```

Finally, the Deployment Diagram Editor generates the deployment diagram from this ADL automatically.

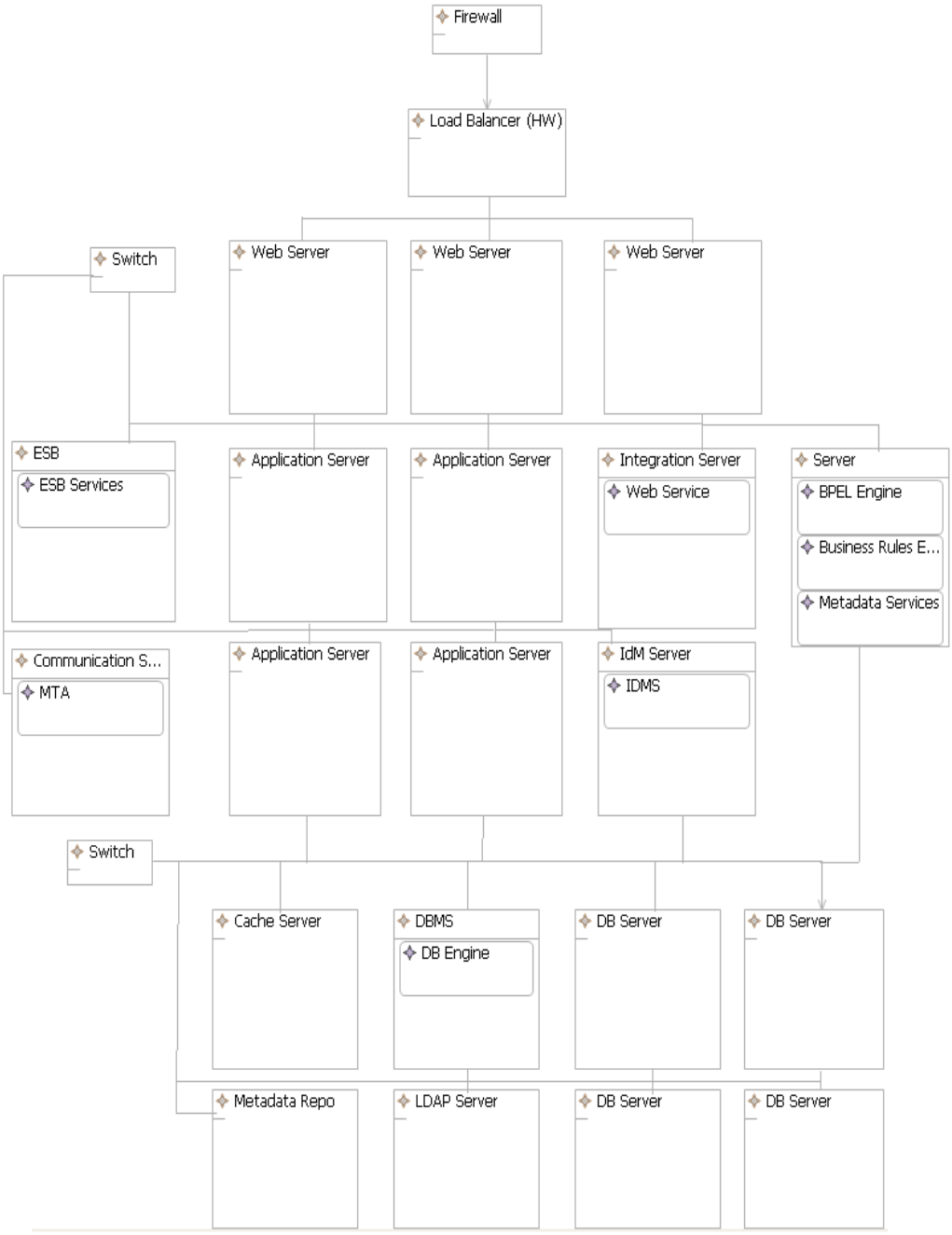


**Figure 34 – Sample Architecture Deployment Diagram**