Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Diplomarbeit Nr. 3317

# Extending an Open Source Enterprise Service Bus for Horizontal Scalability Support

Frederik Festi

| | |
|---|---|
| **Course of Study:** | Computer Science |
| **Examiner:** | Prof. Dr. Frank Leymann |
| **Supervisor:** | Steve Strauch |
| **Commenced:** | April 16, 2012 |
| **Completed:** | October 16, 2012 |
| **CR-Classification:** | C.2.4, D.2.11, H.3.4 |

## Abstract

The Platform as a Service (PaaS) scheme within the Cloud Computing paradigm aims to provide a platform for service providers to deploy and host internet-scale applications. It provides the underlying resources and eases their management, provides integration support, data access and authentication as building blocks and orchestration for Service-oriented Architectures as services are often composites of other services. Two main parts of cloud computing are dynamically scaling resources which adapt to changes in demand and multi-tenancy support to isolate different customers and achieve economy of scale.

This thesis takes an multi-tenancy extension to an open source Enterprise Service Bus (ESB) and adds support for horizontal scalability. First two scalability scenarios a examined for pros and cons and possible solutions and their challenges. Then we specify requirements and design and implement a solution with allows the ESB to scale out and add and remove instance based on performance data gathered and distribute incoming request among them.

# Contents

# List of Figures

# List of Tables

# List of Listings

# 1 Introduction

With markets changing faster and faster and forcing businesses to adapt their processes Cloud computing is the new paradigm promising computation, storage, and application hosting services offering Service Level Agreement (SLA)-backed performance and uptime promises for their services with a "utility" pricing model where customers are charged based on their utilization of computational resources, storage, and transfer of data [BBG11].

The EU project 4CaaSt [4Ca] aims to create an advanced Platform-as-a-Service (PaaS) Cloud platform, which supports the optimized and elastic hosting of composite Internet-scale multi-tier applications. One of the major features of the Cloud is the ability the dynamically scale its resources to changes in traffic and demand. This greatly impacts the problems of over- and underprovisioning and reduces hardware costs. Working with the extension to Apache ServiceMix [SMXa], an open source Enterprise Service Bus (ESB), JBI Multi-tenancy Multi-container Support (JBIMulti2) by Dominik Muhler [Muh11], which adds multi-tenant functionality, this thesis aims to add horizontal scalability support and elasticity. This adds another building block to the complex software system which makes up a PaaS Platform.

## 1.1 Scope of Work

The goal of this thesis is specify, design and implement a solution to facilitate horizontal scalability. This work is based on the ServiceMix extension and multi-tenant management application developed by Dominik Muhler [Muh11]. His application uses the following Technologies: Apache ServiceMix 4.3 [SMXa] an open source ESB, JOnAS an open source Open Services Gateway initiative (OSGi) Enterprise Server [JOn] and the PostgreSQL [PSQ] an object-relational database management system.

Dominik Muhler outlines two clustering scenarios which have to be evaluated. With the knowledge gained an extension to the current system has to be designed and implemented. Monitoring information from Apache ServiceMix shall be used to control dynamic load balancing of incoming request. Elastic provisioning of resources from the underlying Infrastructure Provider has to be enabled as well as managing configuration and deployment of multiple ServiceMix instances and their components.

## 1.2 Outline

This section gives a brief overview of the content of the following chapters.

- **Fundamentals, Chapter 2**—In the beginning an overview of fundamental topics which form the basis of this thesis is given as seen in common literature of each topic. The chapter covers Service-Oriented Architecture (SOA), ESB, Cloud computing and load balancing.
- **Related Works, Chapter 3**—Secondly three other concepts considering load balancing and scalability have been examined.
- **Investigation, Chapter 4**—The two scenarios as defined by Dominik Muhler in his Diploma Thesis "Extending an Open Source Enterprise Service Bus for Multi-Tenancy Support Focusing on Administration and Management" are examined looking at pros and cons and technological challenges required for their implementation.
- **Requirements , Chapter 5**—We formalize the requirements for the application to facilitate horizontal scalability with lessons learned from the previous investigation.
- **Design, Chapter 6**—Gives an overview of the architecture as well as the specifics of the components which satisfy the requirements.
- **Implementation and Evaluation, Chapter 7**—Describes challenges which arose during implementation as well as limitation of the prototype.
- **Outcome and Future Work, Chapter 8**—The last chapter summarizes the work done in this thesis and suggest further possible extensions as well as research topics related to horizontal scalability.

## List of Abbreviations

The following list contains abbreviations used in this document. Full names by convention not valid or not used anymore are marked as deprecated.

| | |
|---|---|
| **API** | Application Programming Interface |
| **Axis2** | Apache eXtensible Interaction System v. 2 |
| **DNS** | Domain Name System |
| **ESB** | Enterprise Service Bus |
| **IaaS** | Infrastructure-as-a-Service |
| **JBI** | Java Business Integration |
| **JBIMulti2** | JBI Multi-tenancy Multi-container Support |
| **JMS** | Java Message Service |
| **JMX** | Java Management Extensions |
| **LAN** | Local Area Network |
| **LDAP** | Lightweight Directory Access Protocol |

| | |
|---|---|
| **MOM** | Message Oriented Middleware |
| **OSGi** | Open Services Gateway initiative *(deprecated)* |
| **PaaS** | Platform-as-a-Service |
| **POJO** | Plain Old Java Object |
| **RMI** | Remote Method Invocation |
| **SaaS** | Software-as-a-Service |
| **SA** | Service Assembly |
| **SLA** | Service Level Agreement |
| **SOA** | Service-Oriented Architecture |
| **SOAP** | Simple Object Access Protocol *(deprecated)* |
| **UDDI** | Universal Description, Discovery and Integration |
| **UUID** | Universally Unique Identifier |
| **VPN** | Virtual Private Network |
| **WS*** | Web Services (Specifications) |
| **WSDL** | Web Services Description Language |

# 2 Fundamentals

This chapter provides the fundamentals this thesis is based on. It gives the reader a brief overview and a point of reference should he seek further knowledge. Topics covered are SOA, ESB, Could computing and Load Balancing.

## 2.1 Service-Oriented Architecture

Service-oriented architecture (SOA) is an architectural style advocating modularisation of services for easier orchestration. Application of SOA allows a business to easily and readily combine and recombine these services to accommodate change and improvements in the business processes. The abstraction of the service interface allow to producers and consumers flexibility in their choice of implementation technologies [Bro08].

The key component is the service which encapsulates a business activity. The service is self contained and produces a specified result. It is a Black Box in that the consumer knows only the interface definition but not the internal workings. The service may be in fact a composition of other services [OG09].

The core of SOA is the SOA-Triangle with it's three methods publish, find and bind. It provides the loose coupling which allow Service reuse and orchestration. A Service Provider uses the publish method to deposit details of his service in a service registry, e.g. Universal Description, Discovery and Integration (UDDI). The requester has then the possibility to find the service given that it lies within the parameters he's looking for. And last the requester uses the service description obtained from the service registry to bind to the actual service. With the introduction of the ESB as message mediator and wide distributed and accepted standards like WS* [WSA] Web services are an established technology. By decoupling service providers from service requesters the ESB plays an important role in any SOA environment. Standing in between the two the ESB functions as mediator, handles the service selection process and mismatches in the interface or security or reliability requirements [FN08].

## 2.2 Enterprise Service Bus

An ESB uses Standards-Based Integration as fundamental concept. Standards Java Message Service (JMS), J2EE, and other are used as components to connect application adapters. These standards-based interfaces and components are put together in a meaningful way that comprises an open-ended pluggable architecture.[Cha04]

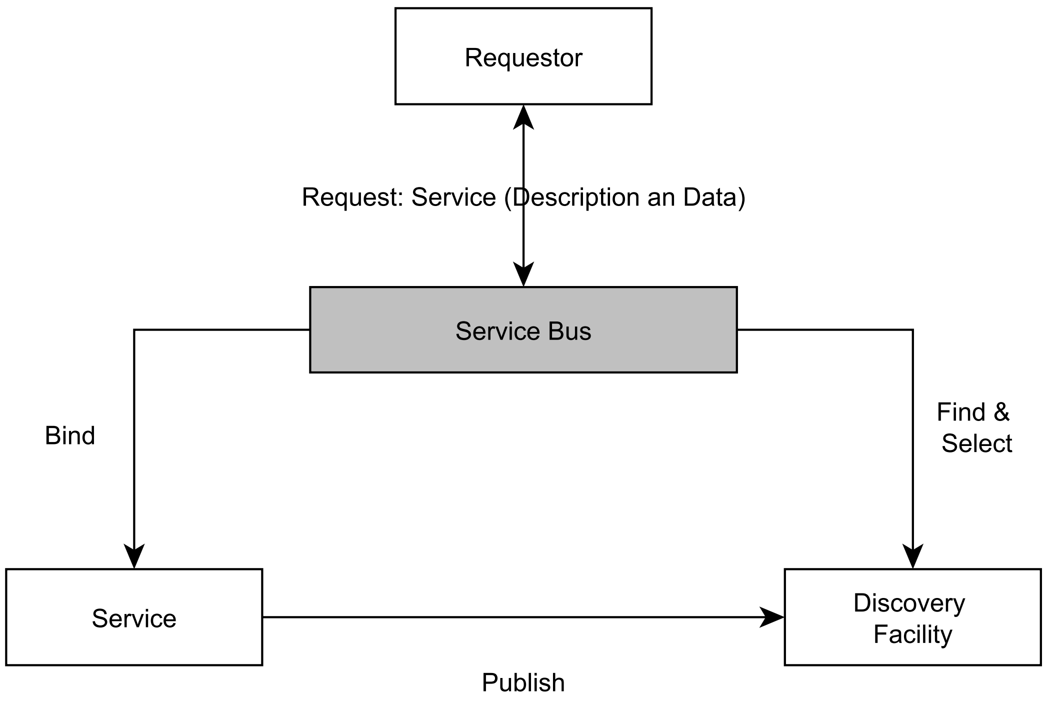


**Figure 2.1:** The SOA Triangle with the Service Bus according to Weerawarana et al. [WCL+06]

An ESB includes a distributed, configurable infrastructure, whose tasks are to provide these core functionalities [FN08]:

- Routing: The ESB acts as a inter-mediator between service requester and service provider.
- Conversion: The ESB handles multiple underlying transport protocols for massage delivery and can convert between them.
- Transformation: The ESB provides to transform messages in between message exchanges should the interfaces of the two participating services not match.
- Aspect-oriented connectivity: Additional functionality like security, logging, management, auditing which add value on top.

Message Oriented Middleware (MOM) stand at the core of every ESB providing reliable messaging via store and forward, message persistence, message transformation and transaction support. Ot is responsible for managing the connection points between multiple clients. Messaging Provider usually have the ability to form clusters which provide load balancing, fault tolerance, and sophisticated routing.

## 2.3 Cloud Computing

Like it's predecessors grid and cluster computing, Cloud computing aims to make computing power a utility but goes beyond the grind and cluster model in that it adds in computing service. Making computing power and services available on-demand in a pay-as-you-go manner as it virtualizes the physical resources behind a 'Cloud' allows businesses and individuals to access them from everywhere in the world at reduced cost and increased flexibility [BBG11]. With the technology of Web 2.0 and easier payment methods like PayPal, with "low-touch, low-margin, low-commitment", have made it possible for a wide audience to use Cloud computing and make it successful where its predecessors have failed to gain broad acceptance [AFG$^+$09].

To make this possible Cloud systems make use of resource pooling to serve multiple customers using a multi-tenant model. Creating a data abstraction while keeping tenants isolated. Adding in rapid elasticity, the ability to scale the available resources to fit the current need. This can happen, to a degree, unilaterally by the consumer without needing human interaction by the service provider side and sometimes even automatically. Combined with broad network access through standard mechanisms to allow heterogeneous applications to communicate and utility for monitoring, reporting and controlling resources these features make the flexible, on-demand model possible[MG11].

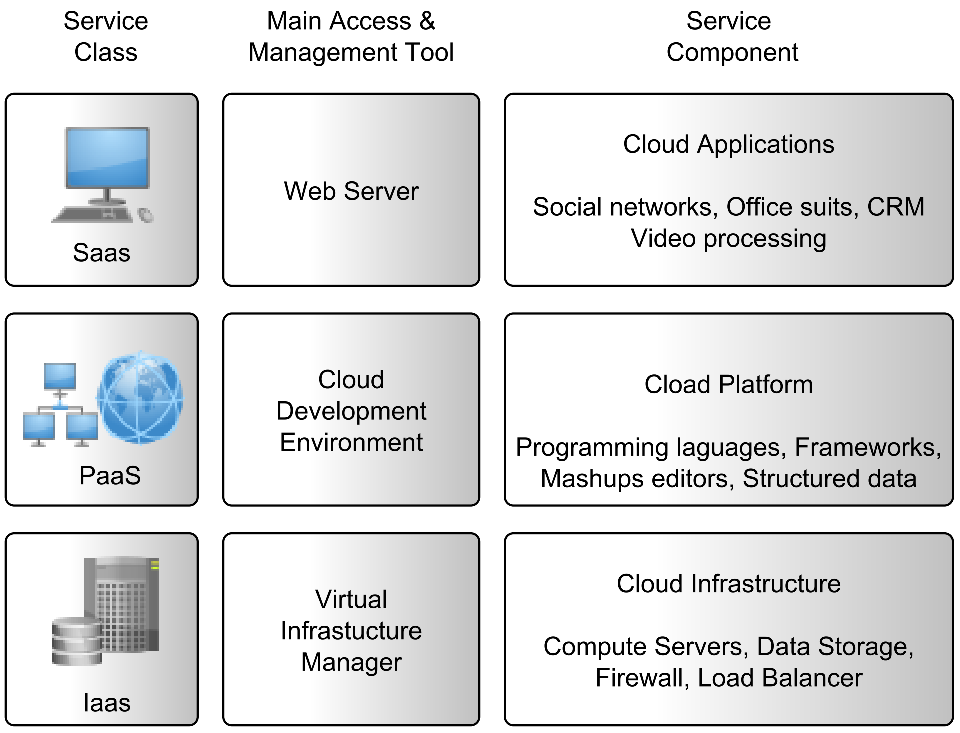

Service Class
Main Access & Management Tool
Service Component
Saas
Web Server
Cloud Applications
Social networks, Office suits, CRM Video processing
PaaS
Cloud Development Environment
Cload Platform
Programming laguages, Frameworks, Mashups editors, Structured data
Iaas
Virtual Infrastucture Manager
Cloud Infrastructure
Compute Servers, Data Storage, Firewall, Load Balancer

**Figure 2.2:** The Cloud Computing Stack according to Buyya et al. [BBG11]

Within the cloud different actors can be distinguished who operate on different level of the cloud computing stack. Infrastructure offer virtualized hardware, like servers or data stores. Service Providers use said infrastructure to deploy their services and make them to Service Users through Internet-based standardized interface [VRMCL09]. Depending on the depth of services that are available we distinguish the following provider types:

- Infrastructure-as-a-Service (IaaS) is base for the rest of the Cloud layers and offers virtualized resources, computation, storage and communication on-demand.
- Platform-as-a-Service (PaaS) adds an higher level abstraction on top the infrastructure level and allows easier development and deployment of applications and services.
- Software-as-a-Service (SaaS) finally offers the deployed services through Web portals to the end consumer removing his need for locally installed programs.

The most relevant part for this thesis is the automatic scaling and load balancing. It is the key part which bring elasticity into the cloud. By scaling automatically to varying load conditions it is a main contributor to the cost reduction offered by Cloud computing [BBG11].

## 2.4 Load Balancing

Load balancing is the distribution of incoming traffic between a group of servers hosting the same application content. This increases overall availability and responsiveness by removing single points of failure and allows to improve server utilization and increases availability. The application server infrastructure can be scaled out, in contrast to scaling up by increasing the power of a single server, which is an easy way to increase performance by simply adding new servers [Sys].

Taking in traffic at one point, one URL, one IP address and redistributing it by an dedicated load balancer who mainpulates the traffic going through it provides three main benefits:

- Flexibility - With load balancing in effect servers can be added and removed at any time. The effect is immediate and allows for the maintenance of any machine, even during peak hours with little or no impact to the site. The other part of this is the intelligent distribution of traffic by using cookies, URL parsing, static and dynamic algorithms, and much more.
- High availability - Load balancing can take any servers in or out of the rotation automatically depending on their status or the result of a periodic health check. Load balancers usually allow themselves to configured for redundancy , usually master and slave, so that they provide a failover in case the primary system goes down.
- Scalability - By distributing load among many servers, load balancing allows to increase the serving power of a site or service by adding more servers. Since many small- to medium-sized servers are usually much less expensive than a few high-end servers this reduces the costs on new hardware purchases. This allows to handle increases on demand by allowing to immediately add new server to handle the traffic. [Bou01]

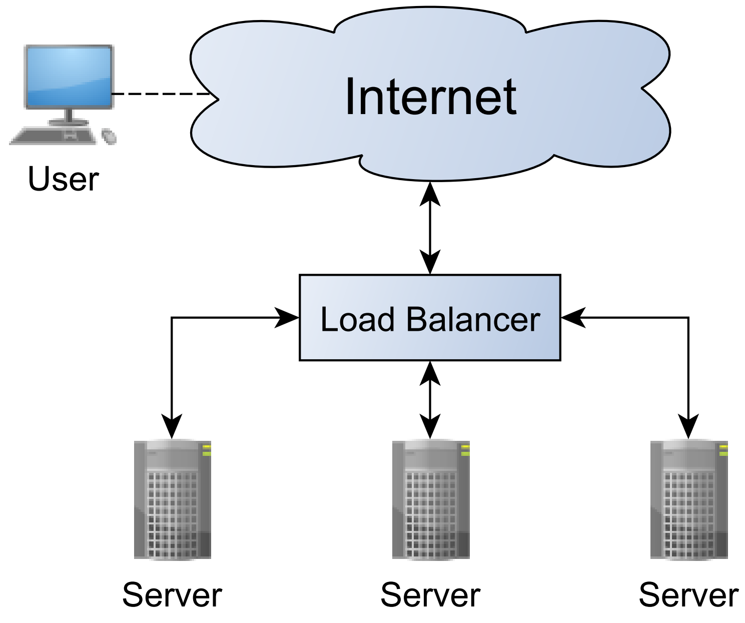


**Figure 2.3:** Simple Load Balancing in [Bou01]

A distinction is often made between Layer 4 or network load balancing and Layer 7 or application load balancing [Cit] [God]. The layers mentioned refer to the OSI networking model.

- Network load balancing: Distributes incoming traffic by network information from the OSI layers 2-4 which includes MAC- and IP-Address and TCP Port. It only takes into consideration the content of a single network package but not the context.
- Application load balancing: an extension of the above as it also has these functionalities but has the abilities to assemble complete request before distribution and thus use of HTTP header, URI, SSL session ID, HTML form data and any other payload data.

### 2.4.1 Common Load Balancing Algorithms

For server selection there are few commonly used load balancing algorithms. These do traffic distribution and work distribution per request not based on the actual load the request creates on the server as a normal load balancer usually doesn't have this information. In the following you find the mostly used algorithms by load balancers [Cis05] [LVS]:

- Round Robin - Goes through every server in a loop treats every one equal. This is not Domain Name System (DNS) round robin so no caching issues arise.
- Least Connections - The load balancer keeps count on each active connection and always routes to the server with the least.

- Weighted Round Robin and Weighted Least Connections - As the above but each server has an assigned weight
- Hashing - Take a part of the incoming connection to create a Hash. This can be the target or source destination or the URL or parts of it.

# 3 Related Works

The following publications investigated Load-Balancing and Scalability issues and solutions that go beyond what usual TCP/HTTP Load-Balancers for dealing with requests against Web Servers do. But each underlying system differs from ours in one or more ways and this project is bound to Apache ServiceMix due to previous works from Stefan Essl and Dominik Muhler [Muh11]. And while they all go into some of the more complex issues regarding Cloud computing and SOA via ESB, none of them include any considerations towards multi-tenancy. Kumar, et al. [KK12] introduces an architecture to deal with and/or leverage the additional complexity that comes with modern state of the art Cloud solutions but the focus lies completely on the infrastructure and services that Amazon provides within their Cloud. Srirama et al[SVoJ10] focuses on the challenges when using SOA in a mobile environment which also includes Load Balancing under these special constraints. Othman et al. [OOS01] use die middleware load balancing mechanisms supported by CORBA ORBs to create a solution for their degrading scalability in distributed systems.

## 3.1 A Generalized Framework for Building Scalable Load Balancing Architectures in the Cloud

The Framework Kumar et al[KK12] suggest is built out of the Cloud components offered by Amazon [AWS]. These components are Amazon Elastic Compute Cloud (Amazon EC2), Amazon S3, Amazon Elastic Block Storage (EBS), Amazon Simple Queue Service (Amazon SQS), Amazon SimpleDB, Amazon Relational Database Service (RDS), Amazon CloudFront and get then managed by the Rightscale management Platform [RsC] to customize and control the underlying cloud components. The adaptation to the required changes can be either done in advance or on the fly.

They define the ideal scalable architecture by the followong characteristics:

- Proportional growth of resources and performance
- Can handle heterogeneity
- Durability
- Cost effectiveness increases with size

By following the guidelines laid out in [Var11] they aim to achieve these characteristics. Mechanisms for automatic recovery are implemented to increase the fault tolerance of the system as taught by the design for failure principle. This also depends on the practice of decoupling to keep an error from affecting more of the system than necessary and ease error handling. Then the elasticity come onto play. Automating deployment and build process allow Software controlled scaling.

Rightscale Cloud Management Service is used for monitoring and administration by adding and removing additional resources to and from the Cloud environment.. For the actual load balancing they either use Amazons Elastic Load Balancing [AEL] or other external load balancers like HAProxy [HAP] or Zeus [Zeu]. Rightscale offers a comparison between the different load balancers in [Adl10].

While they go beyond the usual load balancing of Web servers and add their view on load balancing in the Cloud they unfortunately make no considerations for multi-tenancy concerns.

## 3.2 Scalable Mobile Web Services Mediation Framework

While [KK12] focusses on Cloud arrangements Srirama et al.[SVoJ10] looked at the application of WS* technologies in a mobile environment. They transform mobile devices like Smartphones into Mobile Hosts which can act as both Service Consumers and Service Providers. Connected Mobile Host in a cellular Network form a Mobile Enterprise which introduces new challenges into the usual SOA architecture. Their Mobile Web Services Mediation Framework (MWSMF) uses a set of technologies to address the following issues they identified as upcoming:

QoS: Due to the nature of mobile networks security plays a major role in data connections and message transfer. Using AES (Advanced Encryption Standard) symmetric encryption and message signing on critical parts can address these issues but special care has to be taken since these introduce non-trivial overhead.

Discovery: With Nodes being dynamic in geospatial position, nodes leaving and joining the network as well as changing operators becomes a common happening. this makes static centralized solutions like the standard UDDI suboptimal. Creating a virtual P2P network they use JXTA [JXT], an open source P2P protocol specification, as a backbone for a distributed registry.

Integration: Unfortunately the solutions for QoS and Discovery in addition to the usually non-compatible data-formats generally found in enterprise networks leave us with serious integration problems. They used the ServiceMix ESB as a Backbone to deal with the integration problems.

Scalability: With integration solved, only capacity overload when too many request arrive remained. To facilitate scaling they deploy their MWSMF into the Cloud using the Amazon EC2 Service [AEC]. Using an Apache HTTP server as load balancer they employed horizontal scalability to split incoming traffic among a set of different MWSMF nodes. Since the nodes are stateless no additional complexity is introduced.

Their scalability approach is sufficient for confirming that their MWSMF is indeed horizontally scalable. But their implementation doesn't take elasticity into account and their choice of load balancer is limiting any automated adjustments to the current worker pool because changing the worker pool would require a restart of the load balancer which adds instability as requests can get lost.

## 3.3 The Design of an Adaptive CORBA Load Balancing Service

To combat degrading performance and reduced responsiveness Othman et al.[OOS01] introduced middleware load balancing mechanisms based on CORBA Object Request Brokers. A design , which can be used by similar distributed object computing middleware, like Java RMI, to offer an adaptive load balancing service effective for distributed systems.

They define a load balancer which manages a group of replicas. Each replica has a Load Monitor which gives data to the load balancer via pushing or polling. The Load Balancer has a Load Analyzer Component to dynamically select load balancing strategies as needed or choose which replica is the targets for an incoming request and a Replica Locater Component which uses the Interceptor Pattern via servant locators to bind clients to identified Replicas.

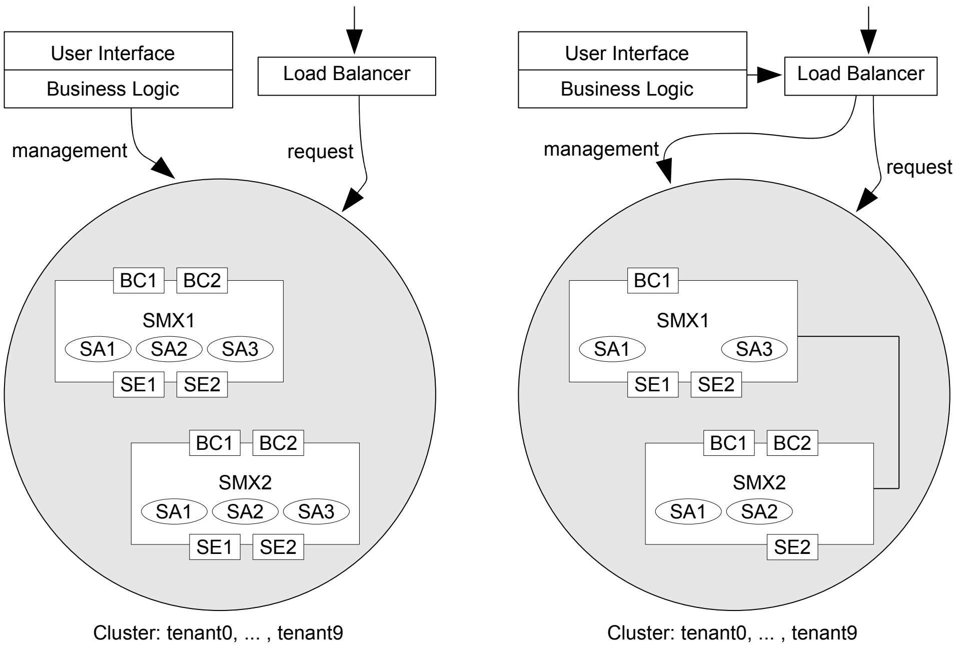
The following patterns are proposed with respect to load balancing:

- The Interceptor Pattern is used to achieve portable load balancing
- The Strategy and Mediator Patterns enhance feedback and control
- The Component Configurator Pattern adds support for modular load balancing
- Common adaptive load balancing hazards are remedied by lowering sampling rates of load statistics
- The Asynchronous Completion Pattern is used to uniquely identify Load Monitors
- The Mediator Pattern helps with integration all the parts of the system

While many problems and corresponding proposed solutions are relevant or provide useful information the technology used differs from the one application mix that is already in use since prior to my involvement in this project.

# 4 Investigation

Dominik Muhler describes in his Diploma Thesis two scenarios for a possible load balancing solutions. One scenario where each ServiceMix instances is separate. They have no connection to each other and have the full set of all Binding Components and Service Engines available deployed. The second scenario has the ServiceMix instances connected by a bridge. With each ServiceMix instance with a reduced amount of deployed components to save resources in case their amount gets too high [Muh11].



(a) Scenario A. (b) Scenario B.

Legend

| | | | |
|---|---|---|---|
| SMX | Apache ServiceMix Instance | SE | Service Engine |
| BC | Binding Component | SA | Service Assembly |

**Figure 4.1:** The two clustering scenarios being evaluated. [Muh11]

## 4.1 Separate Instances

In this Scenario each ServiceMix instance resides on a separate Server with no connection to or knowledge of the other instances of ServiceMix on other servers. With no connection and the possibility of a composite service or orchestration using any possible other service the deployment of every service and component on every server is necessary in this case. If we create tenant groups were the components are varied we can manage a different server pool per group were we can omit components not available for a certain group. But once the number of components of one group exceeds the capacity there is no way to distribute the components between servers without make servers aware of each other and creating a bridge between them.

Using tenant information for routing request to servers would not be necessary when all tenants are treated equal. When every service and component is available on every server it doesn't matter where a tenant specific request ends up. But if we want to manage different groups of tenants , e.g. regular and premium, the load balancer needs to have layer 7 processing to be able to get access to the tenant ID contained in the meta data of the request message. This increases the amount of resources needed per message but if we look at the HAProxy load balancer, common use only uses 5 CPU for user(non-kernel)-space code execution where all layer 7 processing is done. So the overall impact of layer 7 processing minuscule [HAP].

In this case the load balancer would be it's own application running on it's own server so it doesn't have to compete with a ServiceMix instance for resources. The load balancer would manage all other ServiceMix instances like thread a pool or multiple thread pools for different tenants.

## 4.2 Interconnected Instances

Apache ServiceMix has a complete instance of Apache ActiveMQ [AAM] running inside. ActiveMQ is a message broker which implements the JMS [Dea03] standard. ActiveMQ can be used, like any full fledged Message Queueing Middleware, to set up a network of message brokers or queues. With this we can create a load balancing mechanism by using the right topology when we connect the different Message Brokers of the ServiceMix instances. Connecting multiple consumers to a queue is one way of achieving load balancing, creating a JMS topic and distributing messages another [Mah04].

We need to dynamically set up a network of brokers as dynamic load balancing requires to add new nodes on the fly as needed. We have the following possibilities which each offers their own challenges. The <`networkConnector`> element is used to connect different brokers. It offers multiple possible solutions which each has their own challenges:

- Creating static `networkConnectors` from out side via Java Management Extensions (JMX): The Broker MBean of ActiveMQ exposes the `addNetworkConnector` method which allows to add `networkConnectors`. Unfortunately it only offers very limited parametrization. Only the URL to connect to can be specified. The name of the `netwok-Connector` defaults to 'localhost' and cannot be changed afterwards. This prevents us from adding more than one `networkConnector`. If we want to use this method changes to the ActiveMQ code would have to be made to make the exposed `addNetworkConnector` more flexible.

- Multicast Connector / Peer Protocol: Both have automatic discovery but both only work over Local Area Network (LAN). To overcome this limitation we would have to create and manage a Virtual Private Network (VPN) of all the servers the load balancer uses.
- LDAP Broker Discovery Mechanism: Allows to read a list of IPs to connect to from a Lightweight Directory Access Protocol (LDAP) server which has to be run and managed.

Figure 4.2 shows the Concentrator Topology proposed in [Fus10]. The first layer is specialized in accepting lots of messages from producers. The second needs to handle only a small number of connections.

Although dynamic propagation is more flexible, it necessitates sending advisory messages throughout the broker network, which the brokers then use to figure out the optimal route in a dynamic way. As you scale up the network, there is a danger that the advisory messages could swamp the traffic in the broker network.

Static propagation requires you to specify routes explicitly, by telling the broker where to forward messages for specific queues and topics (you can use pattern matching). In this case, you can configure the brokers to disable advisory messages altogether, which eliminates the scalability problems associated with advisory messages.

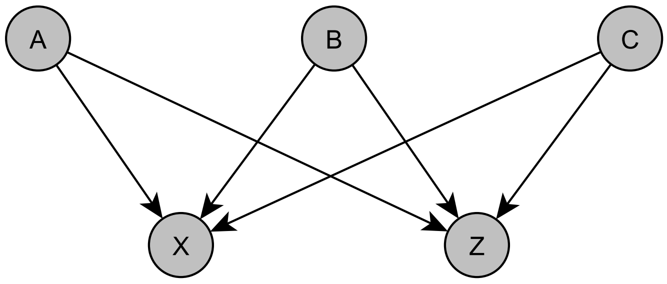


**Figure 4.2:** Concentrator Topology pictured in [Fus10]

ServiceMix ships with a Cluster Engine which uses JMS and per default the supplied ActiveMQ to automatically cluster endpoints and connect multiple ServiceMix instances. Advertised features are [SMXb]:

- Transparent remoting
- Rollback and redelivery when a JBI exchange fails
- Load balancing among JBI containers able to handle a given exchange
- Pause new exchanges processing when the number of concurrently processed messages reach a given threshold

The Cluster Engine is part of the JBI package though which means it cannot be used for all the non-JBI components of ServiceMix. The use of the Service Engine also force the tenants to build their components specifically clustered endpoints. Alternatively the deployment component of the JBIMulti2 Application would have to be extended to change the endpoint definitions of deployed Service Assemblies (SA's) automatically.

```
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:ex="http://example.com/cluster"

...

  <file:poller id="filePoller" service="ex:file-poller" endpoint="endpoint"
   targetService="ex:payload-receiver" file="/tmp/incomingXML" />

  <bean class="org.apache.servicemix.jbi.cluster.engine.OsgiSimpleClusterRegistration">
    <property name="endpoint" ref="filePoller"/>
  </bean>
  <bean class="org.apache.servicemix.common.osgi.EndpointExporter" />
</beans>
```

**Listing 4.1:** A JBI endpoint registered for clustering packaged via OSGi as seen at [SMXc].

## 4.3 Conclusion

The second scenario offers a greater flexibility by being able to distribute components between the different ServiceMix instance but greatly increases the management overhead. The best way to implement the second scenario would be to make use of the built-in JBI clustering in ServiceMix, but this would limit the load balancing to the use of JBI components and SAs. But with ServiceMix 4 the JBI components are already declared depreciated by the developers and they suggest not to use JBI for new projects [SMXd]. While the developers will still include JBI legacy support in upcoming versions [Nod10] it doesn't look like a technology for the future in ServiceMix. And even outside of ServiceMix, JBIs biggest proponent, SUN Microsystems, has since been bought by Oracle and the 2 status on the JBI 2 specification on Java Community Process Web page is set to 'withdrawn' [JBI]. Creating a similar cluster engine for other technologies in ServiceMix would exceed the time frame of this thesis so we will focus on the first scenario to build a basic and simple solution which can be used as a basis for further extensions.

# 5 Requirements

This chapter describes the requirements needed for an adaptive load balancer following the first scenario described in the previous section (see 4.1). After giving an overview of System and how they are connected the requirements of each major component is described in detail.

## 5.1 System Overview

An dynamic load balancing application for both distributing incoming requests and managing the available resource. The application should be run on it's own server without an instance of ServiceMix to have the maximum amount of resources available.

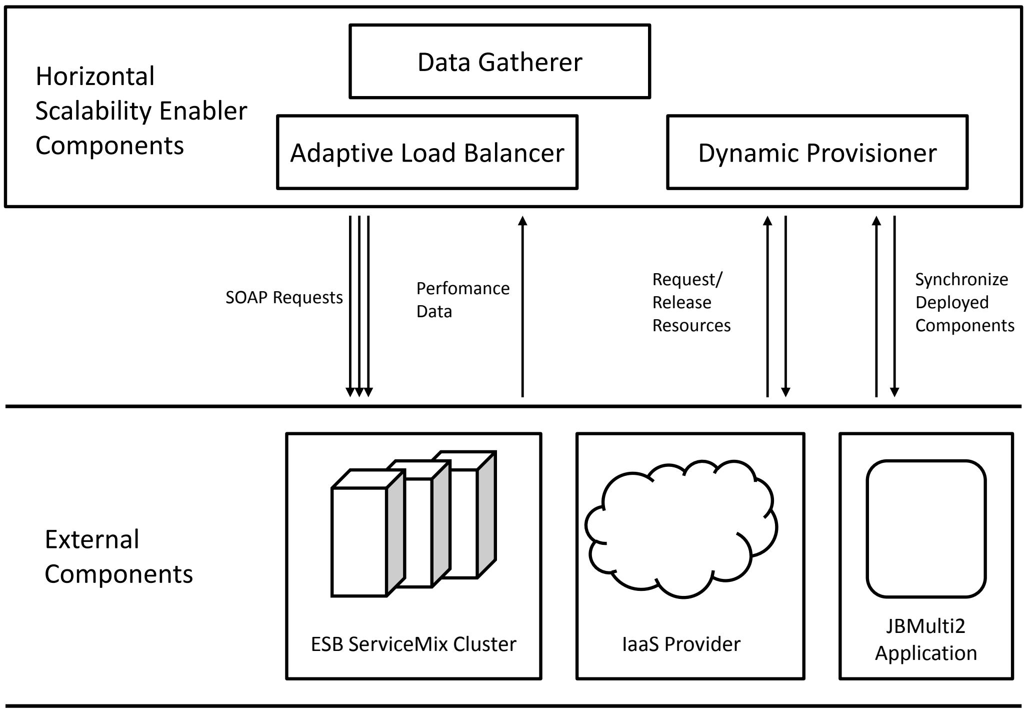


Figure 5.1: System Overview

## 5.2 Utilization-Data Gatherer

There needs to be an interface to connect to each server to gather the current utilization-data. The load balancer uses this data as basis to decide weather new resources are needed or current resources can be released. A class should aggregate this data for easy access of the adaptive load balancer. The data to be gathered must include:

- Memory use of the ServiceMix instance running on the server
- CPU use of the ServiceMix instance running on the server

The information retrieval must happen frequently enough to be up to date and allow an acceptable response time when deploying new resources while not imposing a significant strain on neither the server the load balancer in running on nor the servers of the ServiceMix instances.

## 5.3 Resource Manager

The instances of ServiceMix are each running on their own server. Currently they are running on virtual machine hosted in the Cloud provide by Flexiant [Flea], but this could be any other Cloud service provider or your own servers. But there needs to be a way to administrate them remotely i.e. it need to be possible start/stop or shut-down/reboot. This is needed to enable the flexibility of the load balancing to allocate and deallocate resources to either save energy or make them available for other purposes, if the servers are physical machines belonging to you. Or to reduce cost if they are rented via pay per use from a cloud provider.

The system needs an interface to place an abstraction level above the Application Programming Interface (API) provided by the Cloud provider so the core component, which monitors load and makes the decisions, can allocate new resources or free up existing resources without being bound to a certain API and is able to handle resources from multiple providers.

Besides starting a new server, a routine ensuring that every software component is installed and running and all connections are configured properly.

- The instance of ServiceMix needs te be running
- All Service Assemblies in the Service Registry DB need to be deployed on ServiceMix
- The Server needs to allow JMX connections

## 5.4 Adaptive Load Balancer

The role of the Adaptive Load Balancer is to distribute incoming request in a flexible manner. A suitable algorithm has to be selected that optimizes the use of the currently available resources. Varying amount of processing power needed for different request need to be taken into account as well as different processing capabilities of the servers running the ESB instances. As the response time of a single ESB instance gets worse when it gets closer to maximum capacity the has to be distributed by not only taking into account the number of request but also actual processing power required and consumed. Additionally, there might be different configurations of the servers running the ServiceMix instances. The possibility of difference in processing power and memory available must also be considered.

When the current pool of servers reaches maximum capacity and new servers are added to the pool or a server gets removed, because demand has decreased the dynamic load balancer needs to be able to add or remove these servers to or from the current pool. This needs to happen with no or minimal impact the rest of the servers and currently running connections.

## 5.5 Administrative Interface

Although the load balancing system should mostly operate automated once deployed there are a few parameter that an administrator must be able to change while the system is running to accommodate for changing circumstances.

### 5.5.1 Use Cases

| Name | **List boundaries** |
|---|---|
| Goal | The administrator wants access the boundaries used for Load balancing and adding Resources. |
| Actor | Administrator |
| Pre-Condition | The system is running. |
| Post-Condition | The boundaries are returned. |
| Post-Condition in Special Case | The system remains unchanged. |
| Normal Case | 1. The Administrator request the boundaries.<br>2. A message with the boundaries is returned |
| Special Cases | 1. The Administrator request the boundaries.<br>2. A message with the boundaries is returned |

**Table 5.1:** Description of Use Case List boundaries.

| Name | **Set boundaries** |
|---|---|
| Goal | The administrator wants change one or more boundaries for Load balancing and/or adding Resources. |
| Actor | Administrator |
| Pre-Condition | The system is running. |
| Post-Condition | The system saves the received boundaries. |
| Post-Condition in Special Case | The system still uses the old boundaries |
| Normal Case | 1. The Administrator sends new values of the boundaries he wants to change.<br>2. The application saves the received data<br>3. The system now uses the new boundaries |
| Special Cases | 1. The entered data is not valid<br>a) The system displays an error message. |

**Table 5.2:** Description of Use Case Set boundaries.

| Name | **List Servers** |
|---|---|
| Goal | The administrator wants a list of all the servers which are currently in use. |
| Actor | Administrator |
| Pre-Condition | The system is running. |
| Post-Condition | The list of servers is return. |
| Post-Condition in Special Case | |
| Normal Case | 1. The Administrator request the list of servers. 2. A message with the list of servers which are currently in use is returned. |
| Special Cases | |

**Table 5.3:** Description of Use Case List Servers.

| Name | **Remove Server** |
|---|---|
| Goal | The administrator wants to remove a server manually. |
| Actor | Administrator |
| Pre-Condition | The system is running. |
| Post-Condition | The system has replaced the selected server with a new one. |
| Post-Condition in Special Case | The system remains unchanged. |
| Normal Case | 1. The Administrator sends a request with the ID of the server which should be removed.<br>2. The system start a new server and adds it to the pool.<br>3. The system removes the selected server from the pool.<br>4. The system return a message confirming the successful removal and containing all information about the removes server |
| Special Cases | 2a. The server does not exist any more.<br>a) The system shows an error message and aborts.<br>2b. All possible servers are in use.<br>a) The system shows an error message and aborts. |

**Table 5.4:** Description of Use Case Remove Server.

## 5.6 Non-functional Requirements

In addition to the functional aspects laid out in the previous parts of this chapter the following non-functional requirements have to be fulfilled as well.

- Robustness - Since the load balancer acts as a gateway to all services on every server it manages great care must be taken to ensure the system doesn't break down under strain.
- Security - Since we're building a distributed application and use many management interfaces and communication channels extra care must be taken to ensure no vulnerabilities opened which allow malicious attacks from outside.
- Performance - Being a possible bottleneck performance is an important factor. The impact of the data gatherer and resource manager on performance need to be minimized to allow for a maximum of resources to be available for serving and distributing incoming request.
- Extensibility - The components of the application shall be loosely coupled to make it easier to extend the application in the future exchange different parts for new or other implementations.
- Ease of Installation - The process to install the software and set up a running system must be documented. This should include a step by step guide.

# 6 Design

This chapter describe the proposed solution to facilitate horizontal scalability using Apache ServiceMix with the multi-tenant management extension developed by Dominik Muhler. First an overview of both inner and external components is given with a brief summary. In the subsequent chapters the different parts are described in detail.

## 6.1 Architecture

The Solution has three main components and uses multiple external as well. A Dynamic Load Balancer is responsible for routing request and reacting to varying degrees of utilization of the managed servers running the ESB instances. For the basic load balancing functionality we use the application level load balancer HAProxy [HAP]. A Dynamic Provisioning Module with an interface to the infrastructure provider which requests new servers as needed and configures them so that they can be added to the server pool of the Dynamic Load Balancer. The Data Gatherer connects to the ServiceMix instances via JMX on the servers and constantly gathers utilization data which is used by the Dynamic Load Balancer and the Dynamic Provisioning Module to make their decisions.

Despite most components being predetermined by the JBIMulti2 solution for multi-tenancy support the proposed design tries to keep the components decoupled as much as possible so single parts can be changed more easily. Figure 6.1 show an overview of the components and their connections.

The application is designed to run autonomous once it's been set up but circumstances may arise where some configuration parameters must be changed. For this reason the administrative interface, laid out by the use cases in Chapter 5 will be implemented as a Web service. We use the Apache Apache eXtensible Interaction System v. 2 (Axis2) [ASFa] Web service engine to implement the Web service and Apache Tomcat[ASFb] as servlet container for deployment.

Internally we manage a pool of servers who run ServiceMix for load balancing. The server pool can have a minimum and a maximum size of servers. These can be changed by sending the corresponding SOAP message to the Web service implementing the administrative interface.

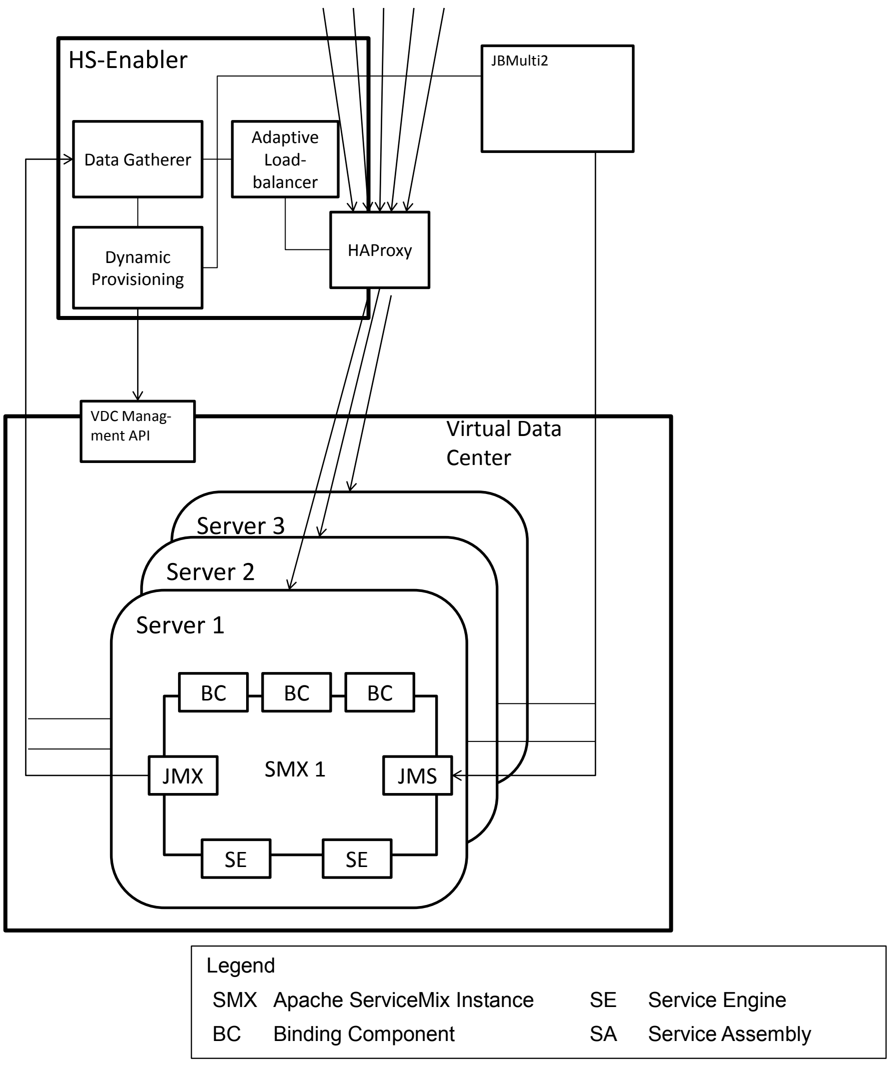


**Figure 6.1:** Architecture Overview

## 6.2 Dynamic Provisioning

The Dynamic Provisioning Module's tasks are monitoring the overall resource utilization and to acquire a new servers when either CPU or memory get close to being fully used and release the servers when utilization sinks again. Before adding servers to the cluster of ESB's the Dynamic Provisioning Module deploys the proper configuration.

We define a simple interface which allows for easy adapter implementation of different IaaS vendors APIs. The interface just define 3 methods:

- createServer
- destroyServer
- getRemainingServers - gives the number of Servers that can still be created.

A adapter has to be written for every IaaS vendor which calls the IaaS's and other administrative APIs to create the server and finish all configuration and startup procedures.

After creating the Server we establish a SSH connection. We use the add exception rule to the firewall. Exception rules are needed for the Data Gatherer te be able to connect to the JMX Management of ServiceMix. We also need to add a exception rule on the server the JBIMulti2 management application is running on allowing the new server to connect to the ActiveMQ JMS topic and be able to receive management messages.

The JBIMulti2 management application defines JBICluster which contain JBIContainer. The JBIContainer represent a equal ServiceMix instance with the same deployed JBIComponents. We use a single JBICluster and add a JBIContainer with every server we create. The current implementation of the JMS messaging between the JBIMulti2 management application and the JMSManagementService deployed on the ServiceMix instances only works locally. We extend the JBIMulti2 application to use it's own ActiveMQ broker. The JMSManagementService then connects to the ActiveMQ instance of the JBIMulti2 application instead of the one running locally in it's ServiceMix instance.

The current implementation of the JBIMulti2 application only synchronizes JBIComponents upon creation of an JBIContainer. But for a complete synchronization we must also deploy any service assemblies currently deployed on the current running ServiceMix instances. We therefore use the Web service interface provided by the to register all service assemblies on the newly created Server.

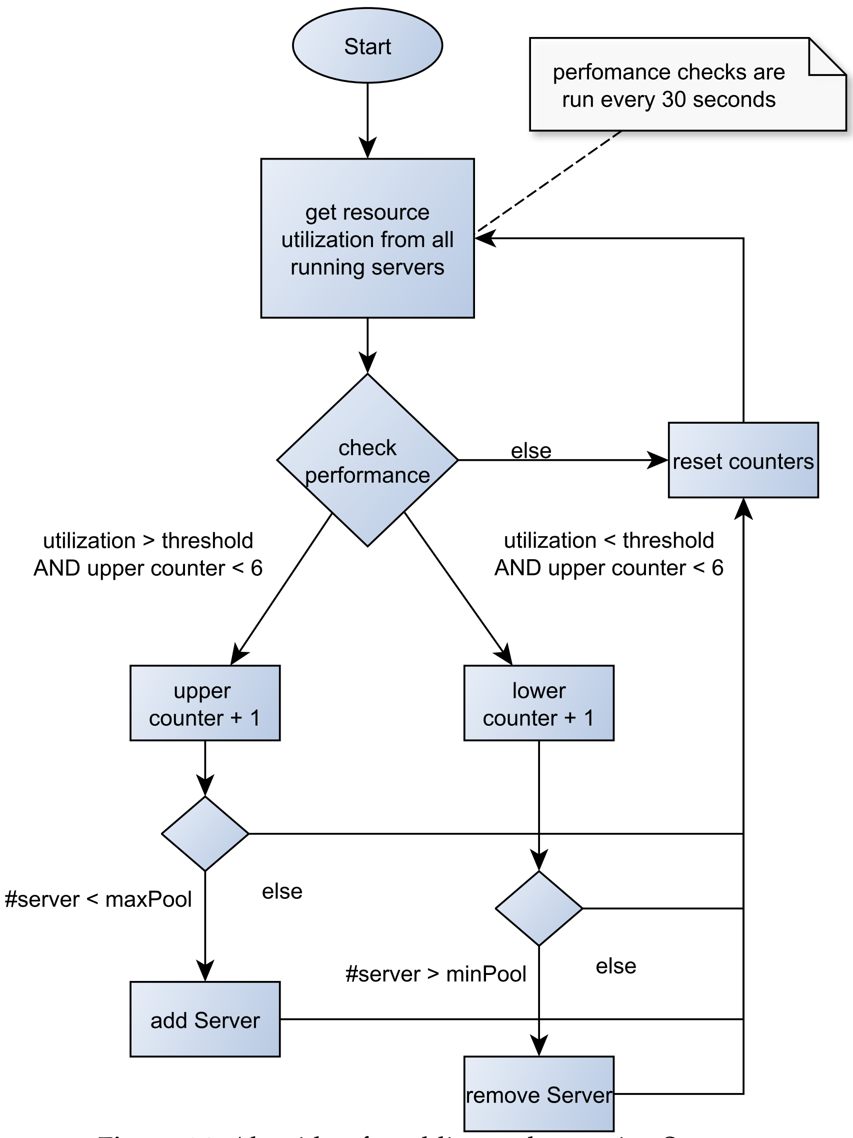
The JMSManagementService gets it's JBIContainer name, the JMS connection URL and the, topic name it connects to via injection at built time. As we need to assign these dynamically we adjust the 'JMSManagementService.xml' file and run the Maven build script. Once we created the specific JMSManagementService we deploy it by using a SSH connection to copy the jar file into the servicemix-home/deploy folder.

For making decisions on adding or removing new Servers we monitor average CPU and memory load in percent over all currently existing servers. We want neither of those values to be at it's maximum for a longer time so we always only take the currently highest into account. We compare this utility percentage against an upper and lower boundary the value of the boundary can be set via the administrative interface. The supplied boundaries are not direct percentages to compare to but are float values representing the amount of free servers, in percent, when action should be taken to optimize the boundaries empirical data will have to be gathered from a running system. For the now we estimate them at the following values:

- Adding Servers: lower boundary = 0.4 ; when 0.4 or less servers in capacity are remaining we add one new server.
- Removing Servers: upper boundary = 1.6 ; when 1.6 or more servers in capacity are available we remove one server.

The Formula to compare % of utilization to the boundaries values:

$$(1.0 - utilization) * \#servers \leq upperBoundary \tag{6.1}$$


Start
perfomance checks are run every 30 seconds
get resource utilization from all running servers
check performance
else
reset counters
utilization > threshold AND upper counter < 6
utilization < threshold AND upper counter < 6
upper counter + 1
lower counter + 1
#server < maxPool
else
#server > minPool
else
add Server
remove Server
**Figure 6.2:** Algorithm for adding and removing Servers

To prevent erratic behaviour periodic checks have to report a boundary violation for at least 3 minutes in a row before action is taken. Figure 6.2 show a flow diagram of the algorithm making the decision weather servers should be added or removed.

## 6.3 Adaptive Load Balancing

Maximizing the number of connections and the sheer data throughput requires OS-specific optimization. We therefore use the already existing load balancer HAProxy which features application level load balancing using an event driven, single process model. It implements the most common load balancing algorithms and exposes management commands via a UNIX socket.

For load balancing we propose to use a weighted load balancing algorithm with HAProxy where we dynamically adjust the weights based on the utilization data received from the ServiceMix instances. When a single server spikes in CPU or memory usage its weight gets reduced their reducing it's strain. And when computing intensive requests have been processed and the weight gets slowly increased back to it's base value again. HAProxy offer two algorithms where this is possible: weighted Round Robin, weighted Least Connections. Which algorithm of the two is superior would have to be tested against benchmarks, e.g the one found at [ESB] with adaptations for multi-tenancy , and will vary with the types of requests which are going to be performed. Our expectation is that the main traffic will be via SOAP over HTTP and the HAProxy documentation recommends round robin in this case because of the relatively short session in http [Tar12].

The weights also allow to accommodate different server capacities. The base value can be adjusted by server. When a standard server start with a weight of 100 a server with a more memory and a faster CPU can have an accordingly increased base weight. Setting the weight to zero prevents a further request from being routed to the server in preparation of shutting the server down to free up resources should the demand decrease enough.

Similar compared to algorithm to add/remove severs is the algorithm adjust weights. The utilization data is specific to each server and we perform checks for each running server. In this case though the upper and lower boundaries, which can be set via the administration interface, correspond directly to a % of utilization. Since the changing of load balancing weights is a lot less time consuming than starting and configuring a Server and ServiceMix instance the enforced sluggishness of the Add/Remove Server algorithm does not have to be reproduce here to that extend.

## 6.4 Data Gathering

For Data gathering we leverage the JMX built into ServiceMix. Each instance of ServiceMix starts it's own MBeanServer. The Data Gathering component is going to connect to every MBeanServer of the every ServiceMix instance we manage. Through the exposed MBeans we query utilization data, CPU and memory use, if the running ServiceMix instances. The Data Gatherer complements a JMX client and makes the gathered data accessible to the Adaptive Load Balancer and Dynamic Provisioning Module

The basic JMX configuration of ServiceMix is done via the 'org.apache.karaf.management.cfg' file found in the 'servicemix-main/etc' folder. Some of the default values, like Port, Username and Password should be changed to ensure security and prevent unauthorized access as JMX allows you to use a plethora of expose management functions via Remote Method Invocation (RMI).

The Data Gatherer queries the utilization Data every 30 seconds. That is low enough not to put much strain on the servers and high enough to give meaningful input. The Data Gatherer only saves the current utilization results. A logging over a extended period of time so not intended as of now.

# 7 Implementation

This chapter list a describes challenges encountered during implementation and certain characteristics of the used software components and technologies.

## 7.1 Dynamic Provisioning

We manage a pool of servers. This pool is different that the pool the load balancer has as servers can exist outside of the load balancing rotation if an Administrator removes him manually in case of manual error handling and diagnosis. To identify each server across multiple possible infrastructure providers we assign each server a Universally Unique Identifier (UUID). The other key attributes are the `containerName` which corresponds to the name attribute of the `JBIContainer` entity in the Configuration Registry of the JBIMulti2 application, the `IPAddress` which has been assigned to the server by the provider, the `ProviderID` which identifies the server at its provider and a `LoadBalancerID`.

The API [Fleb] Flexiant provides to manage their infrastructure is exposed as a Web Service using SOAP over HTTP to communicate. Using the tools provided the Eclipse IDE for Java EE Developers we created a fully functional, easy-to-use, Plain Old Java Object (POJO) Web Service Client by simply importing the Web Services Description Language (WSDL) for the FlexiScale API. The important functions provided from the API are `CreateServer` which is called with a reference to a saved image were ServiceMix is already installed. `StopServer` and `DestroyServer` are used to release resources.

## 7.2 JMX

To successful create a JMX connection to a remote application a few settings have to be ensured. Creating a port exception in the firewall on the target Linux virtual machine running ServiceMix is pretty obvious. But there are some settings not quite as intuitive. My thanks go to Jason Sherman who presents the necessary information concise manner [She12]. The JMX configuration for Apache ServiceMix is located at `{servicemix-home}/etc/org.apache.karaf.management.cfg`

The standard port for the JMX connection is tcp:1099 which can be changed in the configuration file, but we also need to open port tcp:44444 to be open for the connection to be successful. The default login and password are `smx` and `smx` respectively for ServiceMix version 4.3.

This is the service address for the JMX service in generic from an a specific example (IP-address has been changed to a generic LAN address).

- Generic: service:jmx:rmi:///jndi/rmi://localhost:$rmiRegistryPort/karaf-${karaf.name}
- Specific: service:jmx:rmi:///jndi/rmi://192.168.100.1:1099/karaf-root

When we start a new virtual machine from Flexiant[Flea] the hosts file in
`etc` lists itself at the IP 127.0.1.1. This won't allow for other to connect to us so we have to change the entry to the IP the Virtual Data Store from Flexiant assigns to the server on creation. The new entry should look like this: '192.168.100.1 ubuntu.amb.flexiant.com ubuntu'.

## 7.3 HAProxy

The only interface provides HAProxy is through an UNIX socket [Tar12]. We use the 'junix-socket' [Koh] package which is a JAVA/JNI library allowing the use of UNIX Domain Sockets released under the Apache 2.0 License. The UNIX Socket API allows us to set the weights on each server used for load balancing. `enable server` and `disable server` allows us to a server in a down for maintenance state removing it from load balancing and status checks. Unfortunately this interface doesn't allow us to remove server completely or reconfigure name or IP-Address of the server. A complete reconfiguration where changes to the configuration file are read only happen with a restart. This is undesirable because it severs currently active connections. To achieve the elasticity we want we have to pre-populate the HAProxy configuration file with sufficient servers only using server names (e.g. Server01, Server01...) and omitting the IP-Address. HAProxy will then map the server names to IP-Addresses using the
`etc`
`hosts` file from Linux. We can edit this file with admin rights and HAProxy will pick up on the changes.

```
127.0.0.1 localhost
# 127.0.1.1 ubuntu.amb.flexiant.com ubuntu
192.168.10.1  ubuntu.amb.flexiant.com ubuntu

# active server
192.168.100.1  server01
192.168.100.2  server02
192.168.100.3  server03
192.168.100.4  server04
192.168.100.5  server05
```

**Listing 7.1:** The hosts file with the VM itself with the registered IP from the infrastructure provider and the current list of active servers for the load balancer. Put together for ease of reading in this thesis. The server list os only present where the load balancer runs.

# 8 Outcome and Future Work

This diploma thesis adds another part to achieve a full fledged PaaS Cloud Platform. We analysed two scenarios for extending the JBIMulti2 multi-tenant extension to Apache ServiceMix and using the knowledge gained to design and implement a solution to enable horizontal scalability. Using the high-performance application level load balancer HAProxy [HAP] to distribute incoming request we manage a dynamic pool of servers which we create and destroy on demand from Flexiant's [Flea] management web service. We use HAProxy's Unix Socket commands to dynamically distribute load between servers based on utilization data gathered from the ServiceMix instances via JMX [JMX].

Due to time constraints extensive performance test against benchmarks could not be included in this thesis. AndroitLogic offers a benchmark [ESB] for test the performance of ESB's but it must be extended to handle the multi-tenancy aspect introduced by the JBIMulti2 application. [AP11] gives a good insight into metrics and architectures for testing ESB performance.

Using saved images to create servers with the ESB has reduced the programmtic need for installation and configuration significantly. But having that option available isn't granted. Investigating a more dynamic way might be the work of a small extension to this horizontal scalability solution.

# Bibliography

[4Ca] 4CaaSt. http://4caast.morfeo-project.org/.

[AAM] Apache ActiveMQ. http://activemq.apache.org/.

[Adl10] B. Adler. Load Balancing in the Cloud: Tools, Tips and Techniques, 2010. http://www.rightscale.com/info_center/white-papers/Load-Balancing-in-the-Cloud.pdf.

[AEC] Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/.

[AEL] Amazon Elastic Load Balancing. http://aws.amazon.com/elasticloadbalancing/.

[AFG$^+$09] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, University of California, Berkeley, Feb 2009. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf.

[AP11] S. P. Ahuja and A. Patel. Enterprise Service Bus: A Performance Evaluation. *Communications and Network*, 3:133–140, 2011.

[ASFa] T. Apache Software Foundation. Apache AXIS2. http://axis.apache.org/axis2/java/core/.

[ASFb] T. Apache Software Foundation. Apache Tomcat. http://tomcat.apache.org/.

[AWS] Amazon Web Services (AWS). http://aws.amazon.com/.

[BBG11] R. Buyya, J. Broberg, and A. Goscinski. *Could Computing - Principles and Paradigms*. John Wiley & Sons, 2011.

[Bou01] T. Bourke. *Server Load Balancing*. O'Reilly, 2001.

[Bro08] P. C. Brown. *Implementing SOA: Total Architecture in Practice*. Addison Wesley Professional, 2008.

[Cha04] D. Chappell. *Enterprise Service Bus: Theory in Practice*. O'Reilly, 2004.

[Cis05] Cisco. Understanding CSM Load Balancing Algorithms, 2005. http://www.cisco.com/en/US/products/hw/modules/ps2706/products_tech_note09186a00801adbde.shtml.

[Cit] Citrix. Advanced load balancing. http://www.citrix.com/products/netscaler-application-delivery-controller/features/app-delivery/advanced-load-balancing.html.

[Dea03] N. Deakin. Java Message Service. Java Cummunity Process, 2003. http://www.jcp.org/en/jsr/detail?id=914.

[ESB] ESB Performance. www.esbperformance.org/.

[Flea] Flexiant Limited. http://www.flexiant.com/your-cloud/hosting-company/.

[Fleb] Flexiant. FlexiScale API. http://www.flexiscale.com/reference/api/.

[FN08] T. Freund and P. Niblett. ESB Interoperability Standards, June 2008. http://public.dhe.ibm.com/software/dw/specs/ws-esb-interop/ESB_Interop_Standards_WP_060208.pdf.

[Fus10] Fuse Source - Clustering Guide, 2010. http://fusesource.com/docs/broker/5.4/clustering/.

[God] S. Godin. Network Load Balancing versus Application Load Balancing. 2009. https://devcentral.f5.com/weblogs/macvittie/archive/2009/09/15/network-application-load-balancing.aspx.

[HAP] HAProxy. http://haproxy.1wt.eu/.

[JBI] Java Community Process - JBI 2. http://jcp.org/en/jsr/detail?id=312.

[JMX] Java Management Extensions (JMX) Specification, version 1.4. http://docs.oracle.com/javase/7/docs/technotes/guides/jmx/JMX_1_4_specification.pdf.

[JOn] OW2 JOnAS. http://jonas.ow2.org/xwiki/bin/view/Main/.

[JXT] JXTA - The Language and Platform Independent Protocol for P2P Networking. http://jxta.kenai.com/.

[KK12] I. P. Kumar and S. Kodukula. A Generalized Framework for Building Scalable Load Balancing Architectures in the Cloud. *(IJCSIT) International Journal of Computer Science and Information Technologies*, Vol. 3 (1):3015 – 3021, 2012.

[Koh] C. Kohlschütter. junixsocket. http://code.google.com/p/junixsocket/.

[LVS] Job Scheduling Algorithms in Linux Virtual Server. http://www.linuxvirtualserver.org/docs/scheduling.html.

[Mah04] Q. H. Mahmoud. *Middleware for Communications*. John Wiley & Sons, 2004.

[MG11] P. Mell and T. Grance. The NIST Definition of Cloud Computing, 2011. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[Muh11] D. Muhler. Extending an Open Source Enterprise Service Bus for Multi-Tenancy Support Focusing on Administration and Management. Master's thesis, Universität Stuttgart, 2011.

[Nod10] G. Nodet. Thoughts about ServiceMix, 2010. `http://gnodet.blogspot.de/2010/12/thoughts-about-servicemix.html` ; About the future of ServiceMix with version 5 by one of it's core developers.

[OG09] T. Open Group. *SOA Source Book*. VA Haren Publishing, Zaltbommel, 2009. `http://www.opengroup.org/soa/source-book/intro/index.htm`.

[OOS01] O. Othman, C. O'Ryan, and D. C. Schmidt. The Design of an Adaptive CORBA Load Balancing Service, 2001.

[PSQ] PostgreSQL. `http://www.postgresql.org/`.

[RsC] Rightscale Cloud Management. `http://www.rightscale.com/solutions/managing-the-cloud/`.

[She12] J. Sheran. ServiceMix: Remote JMX Connection, 2012. `http://jason-sherman.blogspot.de/2012/03/servicemix-remote-jmx-connection.html`.

[SMXa] Apache ServiceMix. `http://servicemix.apache.org/`.

[SMXb] Apache ServiceMix Clustering - Users Guide - Clustering. `http://servicemix.apache.org/SMX4NMR/13-clustering.html`.

[SMXc] Four things you need to know about the new JBI cluster engine in ServiceMix 4. `http://trenaman.blogspot.de/2010/04/four-things-you-need-to-know-about-new.html`.

[SMXd] ServiceMix - Technology Selection guide. `http://servicemix.apache.org/docs/4.4.x/architecture/technology-selection.html`.

[SVoJ10] S. Srirama, E. Vainikko, V. Šor, and M. Jarke. Scalable Mobile Web Services Mediation Framework. In *Fifth International Conference on Internet and Web Applications and Services*, 2010.

[Sys] C. Systems. Load Balancing. `http://www.citrix.com/lang/English/lp/lp_2308995.asp`.

[Tar12] W. Tarreau. HAProxy - Configuration Manual, 2012. `http://cbonte.github.com/haproxy-dconv/configuration-1.5.html#5.1`.

[Var11] J. Varia. Architecting for the cloud: Best practices, 2011. `http://d36cz9buwru1tt.cloudfront.net/AWS_Cloud_Best_Practices.pdf`.

[VRMCL09] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, Volume 39, Number 1:50–55, 2009. `ftp://doc.nit.ac.ir/cee/jazayeri/research%20method/a%20break%20in%20the%20clouds%20towards%20a%20cloud%20definition.pdf`.

[WCL$^+$06] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall International, 2006.

[WSA] Web Services Architecture. `http://www.w3.org/TR/ws-arch/`.

[Zeu] Zeus Load Balancer. `http://www.xtrahost.co.uk/complex/zeusloadbalancer.html`.

All links were last followed on October 15, 2012.

## Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

Stuttgart, October 16, 2012 ______________________

(Frederik Festi)