Table of Contents

1. Project Overview ........................................................................................................... 6
   1.1 Purpose .......................................................................................................................... 6
   1.2 Current State .................................................................................................................. 6
   1.3 UCLA ESB Benefits ...................................................................................................... 6
       1.3.1 Campus Use case 1 – Efficient Information flow .................................................. 6
       1.3.2 Campus Use case 2 – Resolve data mismatches .................................................... 7
       1.3.3 Campus Use case 3 – Address technology differences ............................................ 7
       1.3.4 Campus Use case 4 – Move toward real-time communications ............................. 7
       1.3.5 Business Benefits ................................................................................................. 7
       1.3.6 IT Benefits ............................................................................................................. 8
       1.3.7 Central common services - management, security, monitoring .............................. 8
   1.4 Architecture Principles ............................................................................................... 9
   1.5 Desired Target State .................................................................................................... 10
       1.5.1 UCLA ESB Solution ............................................................................................ 11
       1.5.2 Service Provider & Consumer model ..................................................................... 12
       1.5.3 PAAS / IAAS model ............................................................................................. 12
       1.5.4 Application Tenancy ............................................................................................ 13
   1.6 Technology Implementation Roadmap ......................................................................... 14
       1.6.1 Short Term ............................................................................................................. 14
       1.6.2 Medium Term ........................................................................................................ 14
       1.6.3 Long Term ............................................................................................................. 14

2. Platform Architecture .................................................................................................... 15
   2.1 Logical Architecture .................................................................................................... 15
       2.1.1 Architecture Components ..................................................................................... 15
           2.1.1.1 Distributed ESB Containers ........................................................................... 15
           2.1.1.2 Message Persistence Layer ............................................................................. 17
           2.1.1.3 Application Stage .......................................................................................... 17
           2.1.1.4 File Transfer Server ....................................................................................... 17
           2.1.1.5 Private Certificate Authority (CA) .................................................................. 17
           2.1.1.6 Identity Management System ........................................................................ 17
           2.1.1.7 Management Application ............................................................................... 18
           2.1.1.8 Monitoring Application .................................................................................. 18
           2.1.1.9 Firewall .......................................................................................................... 18
           2.1.1.10 Load Balancer ............................................................................................. 18
       2.1.2 Software Stack ......................................................................................................... 18
   2.2 Physical Architecture ................................................................................................. 18
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.2</td>
<td>Physical Architecture Description</td>
<td>20</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Dynamic Scaling</td>
<td>20</td>
</tr>
<tr>
<td>3.</td>
<td>Deployment Architecture</td>
<td>21</td>
</tr>
<tr>
<td>3.1</td>
<td>Application Development Environments</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>Deployment Mode</td>
<td>22</td>
</tr>
<tr>
<td>3.3</td>
<td>Virtual Machines</td>
<td>23</td>
</tr>
<tr>
<td>3.3.1</td>
<td>VM Prerequisites</td>
<td>24</td>
</tr>
<tr>
<td>3.4</td>
<td>Memory Allocations for Application Containers</td>
<td>25</td>
</tr>
<tr>
<td>3.5</td>
<td>File System Structure</td>
<td>25</td>
</tr>
<tr>
<td>4.</td>
<td>Application Deployment Architecture</td>
<td>27</td>
</tr>
<tr>
<td>4.1</td>
<td>Application Model</td>
<td>27</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Application Name &amp; Key</td>
<td>27</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Feature Requirements</td>
<td>27</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Access Requirements (Transports)</td>
<td>27</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Security Requirements</td>
<td>28</td>
</tr>
<tr>
<td>4.1.5</td>
<td>Capacity Requirements</td>
<td>28</td>
</tr>
<tr>
<td>4.1.6</td>
<td>Message Persistence</td>
<td>28</td>
</tr>
<tr>
<td>4.2</td>
<td>Application Best Practices</td>
<td>30</td>
</tr>
<tr>
<td>4.3</td>
<td>Deployment from ITServices ALM process</td>
<td>33</td>
</tr>
<tr>
<td>4.4</td>
<td>Application Configuration</td>
<td>34</td>
</tr>
<tr>
<td>4.5</td>
<td>Application Deployment</td>
<td>35</td>
</tr>
<tr>
<td>5.</td>
<td>Security Architecture</td>
<td>36</td>
</tr>
<tr>
<td>5.1</td>
<td>Application Security</td>
<td>36</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Service Management</td>
<td>36</td>
</tr>
<tr>
<td>5.1.2</td>
<td>IP Filtering</td>
<td>38</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Transport and Message level security</td>
<td>38</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Real-time service security</td>
<td>38</td>
</tr>
<tr>
<td>5.1.4.1</td>
<td>Ports</td>
<td>38</td>
</tr>
<tr>
<td>5.1.4.2</td>
<td>PKI</td>
<td>39</td>
</tr>
<tr>
<td>5.1.4.3</td>
<td>Privacy</td>
<td>39</td>
</tr>
<tr>
<td>5.1.4.4</td>
<td>Non-repudiation</td>
<td>39</td>
</tr>
<tr>
<td>5.1.4.5</td>
<td>Authentication</td>
<td>39</td>
</tr>
<tr>
<td>5.1.4.6</td>
<td>Authorization</td>
<td>40</td>
</tr>
<tr>
<td>5.1.5</td>
<td>Batch service security</td>
<td>40</td>
</tr>
<tr>
<td>5.1.5.1</td>
<td>Privacy</td>
<td>40</td>
</tr>
<tr>
<td>5.1.5.2</td>
<td>Authentication</td>
<td>40</td>
</tr>
<tr>
<td>5.1.6</td>
<td>Messaging or MQ security</td>
<td>40</td>
</tr>
</tbody>
</table>
5.2 Platform Security .......................................................... 41

6. Monitoring ........................................................................ 42
   6.1 Monitoring Solution .................................................. 42
     6.1.1 JON ........................................................................ 42
     6.1.2 Nagios ................................................................. 44
   6.2 Log Monitoring .......................................................... 44
   6.3 Fuse Management Console ......................................... 45

7. Appendix A- Abbreviations & Acronyms ...................... 49

8. Appendix B- References ................................................. 50

9. Appendix D- ESB at ITServices ..................................... 51

10. Appendix E- Open Items ............................................. 52
    10.1.1 Disaster Recovery ................................................ 52
    10.1.2 Campus development environment ...................... 52
    10.1.3 Identity management system .............................. 52
    10.1.4 ALM process ..................................................... 52
    10.1.5 JON Monitoring solution DBMS requirement ....... 53
    10.1.6 Global File System (GFS) .................................... 53

Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/22/2013</td>
<td>1</td>
<td>Herman D’Costa</td>
<td>First draft with High Level Architecture</td>
</tr>
<tr>
<td>3/20/2013</td>
<td>2</td>
<td>Herman D’Costa</td>
<td>Addressed reviewers feedback, added project business overview, security architecture details</td>
</tr>
<tr>
<td>5/1/2013</td>
<td>3</td>
<td>Herman D’Costa</td>
<td>Updated esb platform service endpoint hostname, 3.3 Virtual Machines, 4.2 Application Best Practices, Added section 5.1.3 Transport and Message level security, 8 Appendix B – References, 10 Appendix E– Open Items</td>
</tr>
<tr>
<td>5/22/2013</td>
<td>4</td>
<td>Andrew Wissmiller</td>
<td>Minor edits</td>
</tr>
<tr>
<td>6/5/2013</td>
<td>5</td>
<td>Herman D’Costa</td>
<td>4.2 Application Best Practices, 5.1.4 Real-time security,</td>
</tr>
</tbody>
</table>
| 5.1.6 Messaging security,  
| 6 Monitoring  
| 8 Appendix B – References,  
| 10 Appendix E– Open Items |
1. Project Overview

1.1 Purpose
This document proposes the technical architecture of an Enterprise Service Bus (ESB) infrastructure platform at UCLA IT Services. It discusses the high and low level architecture and design considerations that have been addressed while building the infrastructure. The project’s wiki page is at https://spaces.ais.ucla.edu/display/esb/. Refer to the wiki page for the business case for an ESB at UCLA and the project plan that lays out a phased approach to achieve the shared platform for multiple campus initiatives. Current status and other happenings are also available at the wiki page.

Section 1, Project Overview introduces the project’s purpose, benefits, principles, target state, objectives and references to other project documents.

Section 2, Platform Architecture describes the main logical and physical architecture of the platform.

Section 3, Deployment Architecture provides details of how the platform is realized within UCLA ITServices infrastructure, application development environments offered, and virtual machine configurations.

Section 4, Application Deployment Architecture goes into how campus applications will interact and relate with the platform, the application model, best practices, and deployment.

Section 5, Security Architecture discusses the high level security considerations of various areas of the platform. It lays the foundation for a management and governance strategy that will be developed over phases.

Section 6, Monitoring depicts the different options for monitoring and receiving notification alerts about the state of the infrastructure and applications deployed in it.

The Appendixes help readers with abbreviations, acronyms and references to come up to speed with prerequisite topics that are outside the scope of detailing in the document. Open items for the ESB architecture are listed.

1.2 Current State

Refer to the business case document at the wiki page that discusses the current state.

1.3 UCLA ESB Benefits

A UCLA ESB platform satisfies the following main use cases. Current ITServices initiatives such as UCPath, BruinBill, IWE & PAMS are already addressing these use cases using an ESB instance.

1.3.1 Campus Use case 1 – Efficient Information flow
- New campus IT initiative involves multiple student departments that need to exchange data between themselves.
• Instead of each department starting an integration project with every other department, resulting in a web of point to point connections, an ESB platform can be leveraged, where each department integrates once with it.

• The data routing and event notifications can be performed in the ESB.

• The departments need to decide on message formats, and then deliver to or receive from the ESB.

1.3.2 Campus Use case 2 – Resolve data mismatches

• Newly contracted cloud vendor application sends real-time message or batch file to a department's existing financial application. The message format is not supported as a few mandatory data elements don’t exist, while others passed are not supported, and still others passed are not in the right format. To change either application is expensive (budgeting resources, time, infrastructure).

• An ESB platform can be leveraged to perform transformations abstracting the effort from either application thus saving expensive intrusive enhancements and time, keeping the current project on track.

• Missing data elements could be filled in (enrichment), by invoking existing services available in the ESB.

• As part of the next enhancement phase of either application, the transformations can be addressed intrusively in the application itself where it makes sense.

1.3.3 Campus Use case 3 – Address technology differences

• A project needs access to data from an organically grown legacy or Mainframe application.

• The ESB can be leveraged to ensure seamless interoperability and provide different standards protocols and technologies along with different invocation, synchronicity, reliability, programming and messaging styles.

1.3.4 Campus Use case 4 – Move toward real-time communications

• The prevalent batch processing mode for campus student, faculty and staff applications be it registration (bruincard, uid), payroll, billing, parking, purchasing, financial, iamucla, results in delays over 1 or more days.

• Latest technologies, disciplines and tools around an ESB platform encourage designing real-time systems.

1.3.5 Business Benefits

• A shared infrastructure helps campus departments avoid expensive investment to build and maintain their own, saving cost, increasing value and improving operating efficiencies.

• Non-functional, common capabilities are available through the platform thus alleviating applications of this burden, and to reduce duplication.

• Once fine grained services are available in the bus, coarse grained composite ones can be created and orchestrated from them.
• The ESB platform increases operational value when included with a SOA governance solution to define, implement and enforce common sets of policies. *(More about SOA governance in the security architecture section 5.1.1).*

• The ESB can be integrated with a BPM/BPEL service, to provide tools and an environment for the definition and execution of business processes.

• The ESB becomes a gateway to instances at other UC campuses to form a network of a UC Bus sharing services & data.

1.3.6 **IT Benefits**

• It provides a layer where software architects integrate applications without writing much code.

• Applications can leverage well known enterprise integration styles or patterns (EIP).

• Campus applications that need to ensure reliable data delivery can use persistency backed Messaging services of the ESB.

1.3.7 **Central common services - management, security, monitoring**

• As the ESB platform is adopted and used by numerous campus applications, a holistic view of the transactions that are exchanged will be necessary. The ESB infrastructure includes a robust centralized solution for monitoring, management, SLA and alert reporting capabilities.

• The campus ESB can be leveraged to de-couple the security model from applications, ensuring the privacy, authenticity, authorization, non-repudiation, and auditing of all messages moving within the ESB and between service-oriented applications.
1.4 Architecture Principles

The guiding architecture principles for the ESB project are as follows.

a) **It is a shared integration platform.**
   
   It is a shared infrastructure platform for UCLA applications and projects that follows the platform/infrastructure as a Service model (PAAS/IAAS). Characteristics of pass/iaas model are listed in section 1.5.3.

   It is recommended that applications are architected for modularity and layered (n tier, for e.g. user interface, integration & middleware, business & data processing). The integration and message processing components of an application are appropriate to be deployed in this platform.

b) **It promotes operational efficiency.**

   It helps reduce the “next day” processing model prevalent with campus applications today by enabling and moving to the real-time model reducing delays in student, faculty and staff information processes.

   It aids the software development lifecycle for application integrations, so that campus departments don’t need to setup their own development and QA environments.
It becomes the platform through which advanced capabilities such as SOA governance and business process management are realized.

c) **It promotes service agility.**
   It promotes more real-time information flows between campus department applications.
   It facilitates a SOA architecture roadmap for campus IT for complex, intensive initiatives such as UCPath, Faculty Information system, and Financial System Replacement.

d) **It promotes service resilience.**
   It supports highly available, fault tolerant deployments and asynchronous communications for better performance.
   It provides reusable/central services around security, management and monitoring to decouple these common concerns from applications.

e) **It promotes data transparency.**
   It is a single location for tracking data movement, in and out. It includes robust monitoring solutions for tracking data, audit, and notification alerts so that applications support can react quickly.

f) **It promotes innovation.**
   The platform promotes IT standards based communications to guide technology trend adoption on campus, for new emerging protocols, formats or styles. Common technology mediation services will be provided.

g) **It simplifies.**
   The platform is to be designed for easy adoption for campus departments and projects by supporting non-disruptive processes.

h) **Security first.**
   In the development of the ESB platform architecture, security is prioritized over features, performance or convenience.

### 1.5 Desired Target State

What is an ESB? In short, it is a **standards based integration and messaging platform** for technologically disparate applications. It is used to enable what is commonly understood as Business-to-Business (B2B) or application-to-application communications, as opposed to Business-to-Consumer (B2C) or end-user communications.
1.5.1 **UCLA ESB Solution**

The UCLA ESB has the following base features.

a) It provides a service container supporting the deployment and hosting of application integration components.

b) It is not an application hosting platform. Core business logic, state, transaction and data processing reside outside the bus in application data centers and server environments. Integrating with the bus facilitates location transparency.

c) The platform is built from industry standard commercial and open source technologies such as (VMware virtual infrastructure, Unix, Java, OSGI, Redhat, Apache), maintaining vendor neutrality as much as possible.

d) It follows the PAAS/IAAS model of application tenancy, isolation, billing, capacity modeling, periodic upgrades & best practices.

e) It provides separate environments to aid integrations development through the software development lifecycle (Dev, Test, QA, Prod).

At UCLA, the ESB infrastructure is hosted at ITServices. Initiatives across campus departments, business applications such as UCPath, student applications such as BruinBill and IWE, other applications such as Faculty Information system (OPUS), IAMUCLA, and future initiatives such as Financial Replacement can expose services through the bus.

Other departments and applications access and consume the services through the bus. These could be UCLA departments or applications at other UCs or external applications provided by vendors in the cloud such as Oracle and HigherOne.

Each application integrates only with and is secured in the bus. The bus mediates and manages real-time and batch service interfaces.

![UCLA ESB Diagram](image)
1.5.2 **Service Provider & Consumer model**

An application can be deployed into the bus as a **service provider** (servicing clients), or as a **service consumer** (invoking services) or both. The bus routes messages between a service consumer and provider enforcing contracts (for e.g. schematic) and security and provides monitoring. In the process of routing, the service bus provides integration capabilities such as transformation, protocol mediation and orchestration that applications can make use and build from.

![Application Consumer Provider Model](image)

**Fig – Application service provider and consumer model**

1.5.3 **PAAS / IAAS model**

The UCLA ESB is a hybrid of a **Platform as a Service (PAAS) and Infrastructure as a Service (IAAS)** from the cloud computing model. The three main categories of Cloud Computing are SaaS, PaaS & IaaS.

**Characteristics adopted from the PAAS model**

- Policy or standards based transport, message and security protocols.
- Platform and tools provider for integration and messaging based on Apache Karaf.
- Building blocks and APIs such as Apache CXF (for web services), MQ (for messaging) and Camel (for integration patterns).
- Platform upgrades

**Characteristics adopted from the IAAS model**
1.5.4 Application Tenancy

The ESB platform hosts multiple applications and ensures application separation in the infrastructure at the process, memory, file system, addressing and security spaces. An application can still communicate with another hosted in the platform over standard integration interfaces.

![Multi Application Tenancy ESB](image_url)

**Fig – Application tenancy in ESB**
1.6 Technology Implementation Roadmap
The following are the technology implementation goals and roadmap for the ESB project.

1.6.1 Short Term
- Provide an integration platform for UCPath application interfaces between Oracle & UCLA. ESB infrastructure to mediate all UCPath interfaces in both directions - batch, real time and ODS.
- Provide a shared integration & messaging platform for UCLA ITServices applications with their external applications.

1.6.2 Medium Term
- Provide robust platform management & monitoring capability
- Provide reusable security services (authentication, authorization, audit)
- Introduce concept of application identities and integrate with existing IDM solution/s
- Consulting services to help teams architect Integration solutions using this platform

1.6.3 Long Term
- Provide a shared integration platform for UCLA department applications
- Provide SOA Governance capabilities
- Provide BPM capabilities
2. Platform Architecture

2.1 Logical Architecture

2.1.1 Architecture Components

2.1.1.1 Distributed ESB Containers
These are collectively the main component of the platform, the actual service bus. A container is at its core a process onto which applications are deployed. It provides a basic set of features around operating system integration, configuration, provisioning, deployment, logging, security and management. The platform next exposes a tool box of foundational capabilities (services, messaging & integration frameworks) that applications can use to architect and develop integration applications and deploy them to the containers.

The services framework enables real-time communications over industry standard web service protocols such as SOAP & REST. It also supports transport and message level security for the services. Apache CXF provides the services framework.
The **messaging framework** facilitates asynchronous communications and reliability using **point-to-point** or queue and **publish-subscribe** or topic channels. Apache MQ provides the messaging framework.

The **integrations framework** bridges the various frameworks together and provides enterprise application integration (EAI) and integration patterns (EIP) capabilities. Apache Camel provides the integrations framework. Some common EIPs are listed in the following table.

**Note** – References for the open source Apache frameworks (CXF, MQ & Camel) are available in the References Appendix.

Campus applications are deployed and access real-time & batch services in the service bus routed through a firewall and hardware load balancer layer. **Real-time services** typically support a medium to high throughput of small to medium size data and are processed in a few seconds, whereas **batch services** support a low frequency of large size data and take few hours or days to be processed. Real-time services deal in a XML format, whereas batch services deal with fixed or delimited UTF-8, XML or a binary format.

<table>
<thead>
<tr>
<th>EIP</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Router</td>
<td>Decouple individual processing steps so that messages can be passed to different filters depending on a set of conditions.</td>
</tr>
<tr>
<td>Content Based Router</td>
<td>The implementation of a single logical function (e.g. a check) is spread across multiple physical systems.</td>
</tr>
<tr>
<td>Message Translator</td>
<td>Systems using different data formats communicate with each other using messaging.</td>
</tr>
<tr>
<td>Request Reply</td>
<td>An application sends a message and gets a response from the receiver.</td>
</tr>
<tr>
<td>Correlation Identifier</td>
<td>For a requestor that has received a reply to know which request this is the reply for.</td>
</tr>
<tr>
<td>Return Address</td>
<td>For a replier to know where to send the reply.</td>
</tr>
<tr>
<td>Message Filter</td>
<td>A component avoids receiving uninteresting messages.</td>
</tr>
<tr>
<td>Content Filter</td>
<td>Simplify dealing with a large message, when interested only in a few data items.</td>
</tr>
<tr>
<td>Recipient List</td>
<td>Route a message to a list of (static or dynamically) specified recipients.</td>
</tr>
<tr>
<td>Splitter</td>
<td>Process a message if it contains multiple elements, each of which may have to be processed in a different way.</td>
</tr>
<tr>
<td>Aggregator</td>
<td>Combine the results of individual, but related messages so that they can be processed as a whole.</td>
</tr>
<tr>
<td>Resequencer</td>
<td>Get a stream of related but out-of-sequence messages back into the correct order.</td>
</tr>
<tr>
<td>Routing Slip</td>
<td>Route a message consecutively through a series of processing steps when the sequence of steps is not known at design-time and may vary for each message.</td>
</tr>
<tr>
<td>Throttler</td>
<td>Throttle messages to ensure that a specific endpoint does not get overloaded, or to not exceed an agreed SLA with some external service.</td>
</tr>
<tr>
<td>Load Balancer</td>
<td>Balance load across a number of endpoints.</td>
</tr>
<tr>
<td>Content Enricher</td>
<td>Communicate with another system if the message originator does not have all the required data items available.</td>
</tr>
<tr>
<td>Event Driven Consumer</td>
<td>An application automatically consumes messages as they become available.</td>
</tr>
</tbody>
</table>
### Message Persistence Layer
Message Brokers used for asynchronous behaviors usually support having a persistence layer for transient messages for reliability. The platform offers a choice to applications between a network file system (NFS) and a database such as MS SQL Server. The communication between the broker and the persistence layer for NFS is TCP/IP and File IO based, and for database is protocol like ODBC or JDBC.

### Application Stage
This is a NFS area for applications to save temporary work files during message processing. Based on requirements an application will be assigned its own exclusive folder.

### File Transfer Server
This component provides secure file transfers across the internet and intranet to the ESB platform for mediating and managing batch services.

The ITServices FX infrastructure from GoAnywhere vendor is an independent Managed File Transfer (MFT) solution for campus departments and applications. The ESB platform provides well defined interfaces with FX and it is adopted as a logical component of the architecture.

### Private Certificate Authority (CA)
The ESB platform service is associated with its own private certificate authority that is hosted in the infrastructure. The private CA is used to issue and sign client certificates only (not server certificates) for remote consumers or clients that access application services in the bus using HTTP over SSL with client authentication (also known as mutual authentication or 2 way SSL). Using a private CA instead of relying on certificates from well-known industry signing authorities enables exclusivity and an extra level of security to real-time communications.

### Identity Management System
This component provides an identity store (commonly backed by a LDAP based directory) that stores end-user and application identities for authentication and authorization. End-user authentication is needed when administrators and operators access consoles (web or ssh). Application authentication is needed in ESB & MQ containers during integration data exchanges. The distributed containers and central management servers can use a framework like Java Authentication & Authorization Service (JAAS) to integrate with an IDM system.
2.1.1.7 Management Application
This component provides management features to centrally manage a network of distributed ESB containers. A persistence mechanism (for e.g. DBMS) to store configurations data, and commonly a web-based GUI for administrators to operate with, forms this component. Configuration and Deployment management features are generally available through this component.

2.1.1.8 Monitoring Application
This component collects runtime data from across the network of distributed ESB containers, collates it, perform calculations and statistics for reports, web GUI console or through notifications (for e.g., Email, SNMP). The monitoring data is typically persisted in a database management system (DBMS). Redhat’s JBoss Operations Network (JON) that provides monitoring features has tight integration and support for the JBoss Fuse esb suite, is a possible solution.

2.1.1.9 Firewall
This component separates and protects ESB platform resources and applications from other networks (UCLA, UC, Cloud Vendors & Internet). The firewall provides the 1st layer of security based on IP address filtering. By default services hosted in the esb are disabled across the firewall. They have to be enabled by configuring white list network/IP ranges in the firewall. The ITServices firewall is leveraged for this component.

2.1.1.10 Load Balancer
This component provides traffic load balancing and high availability capability to services hosted in the ESB platform. The load balancer is configured to route incoming real-time service requests for the esb service host name (webservices.it.ucla.edu) to esb containers. Separate routing rules address esb administration web consoles. The ITServices CSS hardware load balancer is leveraged for this purpose.

2.2 Physical Architecture

2.2.1 Software Stack
Redhat’s JBoss Fuse is a commercially supported open source enterprise product that has been chosen as the software application for the ESB platform (pending any future evaluation of ESB software & vendors based on university business needs). It is built on other open source technologies & industry standards, Apache Servicemix, OSGi & Java. Servicemix is a package distribution of other Apache and third party open source frameworks that essentially make up an ESB.

In Fuse ESB, each container is a java process with the Apache Karaf framework, onto which applications are deployed. “Apache Karaf is a small OSGi based runtime which provides a lightweight container onto which various components and applications can be deployed.” It provides a base set of container features such as hot deployment, dynamic configuration, logging system, provisioning, native OS integration, extensible shell console, remote SSH access, security framework based on JAAS and some management features.”

The Fuse Fabric extension in the base Redhat JBoss Fuse distribution provides the management capability that gives ease of use for operations of the ESB platform. The management server is built on top of Apache Zookeeper open source technology. FuseManagementConsole distribution is used for the web console which is a frontend to the Fabric. It provides configuration, deployment, software patch management, and basic monitoring features for containers, jvm, mq brokers, camel routes and cxf services.
ESB Physical Architecture

**DISTRIBUTED ESB CONTAINERS**

- VM1
  - APP1
  - APP2
  - APP3
- VM2
  - APP2
  - APP1
  - APP3
- VM3
- VM4

**CENTRAL MANAGEMENT SERVERS**

- VM1
  - Fuse Management Console & Fabric
  - Fuse Fabric
- VM2
  - Fuse Management Console & Fabric

**Message Persistence Layer**

- Network File System
- Message Persistence DBMS
- MS SQL Server

**IDM (LDAP)**

**Private Certificate Authority (CA)**

**Fig – ESB Physical Architecture in Production Environment**
2.2.2 Physical Architecture Description

The ESB infrastructure runs atop the ITServices Redhat Linux Virtual Machine (VM) infrastructure. The physical infrastructure primarily comprises of 2 VM clusters, the Distributed ESB containers, where application web services, integration solutions and messaging brokers are deployed, and the Central Management Servers, where management and monitoring solutions are deployed to operate the platform. Each VM hosts one or more containers that are provisioned for applications. The following table lists the default software image requirement for a VM delivered by IS for the ESB platform.

For the management servers cluster, the Fuse Fabric technology requires an odd number of management service instances running due to its internal quorum voting algorithm. Three instances will be deployed for fault tolerance across two VMs.

An application’s service interfaces are typically deployed to two (or more) containers across VM hosts for fault tolerance and performance load balancing. The application code binaries and configurations including ESB dependency libraries such as Apache Karaf, XF, MQ & Camel are remotely deployed to the containers from the central servers.

2.2.3 Dynamic Scaling

As applications are added to the platform and containers provisioned with computing resources allocated based on capacity requirements, the platform needs to be elastic and scalable. The VM infrastructure has the capability to dynamically and relatively quickly adjust a VM’s memory, CPU and hard disk configuration.

VMs can also be added to the ESB containers cluster and easily managed from the central servers. Memory configurations can be specified while provisioning containers remotely from the central servers that slice the VM’s memory among application containers. A VM’s CPU processing power cannot be deterministically sliced among the containers though.
3. Deployment Architecture

3.1 Application Development Environments

The ESB infrastructure makes five environments available for application development lifecycle management (SDLC / ALM) and platform operations use cases. Development, Unit Test, Functional Test, System Integration Test (SIT), Quality Assurance (QA), Staging, Performance Test activities can all be performed in appropriate environments. Having these five environments enhances flexibility and agility for concurrent project lifecycles among application teams. Attributes of each environment are summarized in the following table.

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Environment Name</th>
<th>Purpose</th>
<th>Infrastructure Class / SLA</th>
<th>Firewall Access</th>
<th>SSH Access to VMs</th>
<th>Load balanced (CSS) ?</th>
<th>Capacity factor (scale 1-5; Performance, Stability, High Availability)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sandbox</td>
<td>Lab environment. IMS team tests patches, upgrades, configuration points before applying the same in other environments</td>
<td>DEV</td>
<td>ITServices</td>
<td>IMS - root; Application teams - none</td>
<td>NO</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Development</td>
<td>Development environment for application teams. Fluid environment. Shared responsibility to keep this environment relatively stable</td>
<td>DEV</td>
<td>ITServices</td>
<td>IMS - root; Application teams - root</td>
<td>NO</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Test</td>
<td>Functional test, PoC, String test, (limited) SIT, Performance/Load test, Security test (appscan). Stable environment.</td>
<td>TEST</td>
<td>ITServices, UCLA, UC networks, Oracle On Demand, Other vendors</td>
<td>IMS - root; Application teams - view</td>
<td>YES</td>
<td>3</td>
</tr>
</tbody>
</table>
### 3.2 Deployment Mode

JBoss Fuse supports the installation of the ESB software suite and deployment of application codes and configurations onto the VM infrastructure in two modes, Centralized Management & Standalone.

In the **Centralized Management** mode (branded Fuse Fabric), one installation and configuration of either JBoss Fuse esb or JBoss Fuse ManagementConsole distributions have to be done on a central server cluster VM. After that, all other central services and distributed containers can then be installed and configured remotely. The central nodes provide operations for a scalable infrastructure where container installations and configurations; application configurations, versioning and deployments; platform software upgrades and patching; and monitoring are performed remotely, easily, increasing repeatability and reducing errors.

In this mode, containers are provisioned by default with a small footprint (software libraries, configuration, and ports) and are scaled up as per the need.

In the **Standalone** mode, each container’s installation and configuration needs to done separately. There is no unified management capability (no central services). All management, monitoring, repeatable configurations, upgrades, patching features have to be custom developed using scripting and standard interfaces provided by Apache Karaf based on OSGI and JMX. JBoss Fuse and JBoss A-MQ distributions are used in this mode.

With the out-of-box benefits of the centralized management mode to operate and manage a scalable infrastructure that has the possibility to grow large, it is the logical choice for the deployment mode. The **caveat to this choice is that it is a relatively recent functional extension to the JBoss Fuse suite, and in the**

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Environment Name</th>
<th>Purpose</th>
<th>Infrastructure Class / SLA</th>
<th>Firewall Access</th>
<th>SSH Access to VMs</th>
<th>Load balanced (CSS)?</th>
<th>Capacity factor (scale 1-5; Performance, Stability, High Availability)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>QA</td>
<td>SIT, UAT, Staging, Training, Performance/ Load test. Stable environment. All tests must be scheduled. Data may be refreshed from time to time.</td>
<td>QA</td>
<td>ITServices, UCLA, UC networks, Oracle On Demand, Other vendors</td>
<td>IMS - view; Application teams - view</td>
<td>YES</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>Production</td>
<td>Production</td>
<td>PROD</td>
<td>ITServices, UCLA, UC networks, Oracle On Demand, Other vendors</td>
<td>IMS - view; Application teams - (limited) view</td>
<td>YES</td>
<td>5</td>
</tr>
</tbody>
</table>
process of gaining maturity. However Redhat is committed to the support and feature enhancements of the Fabric extension.

### 3.3 Virtual Machines

The ESB platform immediately needs to support the real-time and batch service mediations for the following four projects (UCPATH, BruinBill, PAMS, and IWE) and in the near term, OPUS and IAMUCLA. Application capacity requirements are analyzed as part of a formal onboarding process.

Keeping in mind the scalability requirement for the platform, the computing resources sheet of the base esb software suite, the flexibility needed for concurrent projects development, the separation required to conduct esb platform R&D and test upgrades and patches without disrupting applications development, the following table proposes the initial configuration for VM computing resources, with emphasis on the need to add computing power as applications are rolled into the platform and to bill or recharge them deterministically based on capacity requirements.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Environment Name</th>
<th>Memory</th>
<th>Hard Disk</th>
<th>CPU Core</th>
<th>Quantity</th>
<th>Software Architecture Component</th>
<th>Server Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sandbox</td>
<td>4 GB</td>
<td>25 GB</td>
<td>2</td>
<td>2</td>
<td>FMC, FAB, JRE7, SPLNK, RHEL</td>
<td>mi-as-d07, mi-as-d08</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ESB, MGMT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Development</td>
<td>6 GB</td>
<td>25 GB</td>
<td>2</td>
<td>1</td>
<td>FAB, FESB, FMQ, JRE7, SPLNK, RHEL</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ESB</td>
<td>mi-as-d11</td>
</tr>
<tr>
<td>3</td>
<td>Test</td>
<td>4 GB</td>
<td>25 GB</td>
<td>2</td>
<td>2</td>
<td>FAB, JRE7, SPLNK, RHEL</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ESB</td>
<td>mi-as-t07, mi-as-t08</td>
</tr>
<tr>
<td>4</td>
<td>Test</td>
<td>6 GB</td>
<td>25 GB</td>
<td>2</td>
<td>1</td>
<td>FMC, FAB, JRE7, HTTPD, SPLNK, RHEL</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MGMT</td>
<td>mi-as-t09</td>
</tr>
<tr>
<td>5</td>
<td>QA</td>
<td>6 GB</td>
<td>25 GB</td>
<td>4</td>
<td>2</td>
<td>FAB, JRE7, SPLNK, RHEL</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ESB</td>
<td>mi-as-q01, mi-as-q02</td>
</tr>
<tr>
<td>6</td>
<td>QA</td>
<td>8 GB</td>
<td>25</td>
<td>4</td>
<td>2</td>
<td>FMC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MGMT</td>
<td>mi-as-q03,</td>
</tr>
</tbody>
</table>
### Software short name

<table>
<thead>
<tr>
<th>Software short name</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMC</td>
<td>JBoss Fuse ManagementConsole 6.x</td>
</tr>
<tr>
<td>FAB</td>
<td>JBoss Fuse Fabric Karaf container 6.x</td>
</tr>
<tr>
<td>FESB</td>
<td>JBoss Fuse 6.x</td>
</tr>
<tr>
<td>FMQ</td>
<td>JBoss A-MQ 6.x</td>
</tr>
<tr>
<td>JRE7</td>
<td>Oracle Java 1.7 update 11 or later</td>
</tr>
<tr>
<td>SPLNK</td>
<td>Splunk</td>
</tr>
<tr>
<td>HTTPD</td>
<td>Apache HTTPD</td>
</tr>
<tr>
<td>RHEL</td>
<td>Redhat Enterprise Linux</td>
</tr>
<tr>
<td>PCA</td>
<td>Private CA software</td>
</tr>
</tbody>
</table>

#### 3.3.1 VM Prerequisites

The following table lists the base software image required for a VM created for the ESB platform. Each VM also needs a local account called fuse with group fusesource and allowed remote write access using ssh to the /usr/local/fuse folder. The local account is used to remotely install and manage the ESB software from central servers.

<table>
<thead>
<tr>
<th>Software</th>
<th>Description</th>
<th>Originating Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redhat Linux</td>
<td>Standard unix operating system</td>
<td>IS</td>
</tr>
<tr>
<td>Oracle Java</td>
<td>Standard distributed platform</td>
<td>ESB</td>
</tr>
</tbody>
</table>
### 3.4 Memory Allocations for Application Containers

The memory defaults for a container provisioned in the platform are as follows.

Min Heap Memory (-Xms) – 128M  
Max Heap Memory (-Xmx) – 512M  
Min Permanent Generation (-XX:PermSize) – 16M  
Max Permanent Generation (-XX:MaxPermSize) – 128M

On review of an application’s capacity requirements, the defaults can be changed. For message broker containers, based on an application’s messaging usage (object sizes, volume & throughput), a memory configuration where the min and max is the same (512M) is suitable to prevent the Java Virtual Machine (JVM) spend CPU cycles resizing, thus optimizing performance.

### 3.5 File System Structure

In each VM, the ESB software is installed under `/usr/local/fuse`. This is known as the root or parent installation. The root installation is a running container itself, needed for management and administration purposes and has a low footprint. Application containers get installed under the `instances` subfolder of the root installation, as shown in the following figure.

<table>
<thead>
<tr>
<th>Software</th>
<th>Description</th>
<th>Originating Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Splunk agent</td>
<td>Log monitoring</td>
<td>ESB</td>
</tr>
<tr>
<td>Puppet agent</td>
<td>Network software &amp; configuration automated deployment</td>
<td>IS</td>
</tr>
<tr>
<td>Nagios agent</td>
<td>Network monitoring</td>
<td>IS</td>
</tr>
</tbody>
</table>

Table – Base software image of an ESB VM
Fig – File system structure in a VM
4. Application Deployment Architecture

4.1 Application Model
An application onboarding process will engage with provider application teams to determine requirements, explain the model and process, terms and conditions of locating in the ESB, and costs.

4.1.1 Application Name & Key
Each provider application deployed in the ESB will have a name or unique key associated (for e.g. ucpath, idm, cb, pams, iwe, iamucla, opus). Configurations and properties are then derived based on the key.

4.1.2 Feature Requirements
Software libraries, configurations and properties that the platform needs to ensure for the application.
- Real-time or web services
- Batch
- Event driven or Messaging
- Transformations
- Enterprise Integration Patterns

4.1.3 Access Requirements (Transports)
Applications identify how their real-time services are accessed, https with basic ssl, https with 2 way ssl or even http.

The platform will register the webservices.it.ucla.edu hostname to service real-time communications. A provider application's address is derived from this DNS hostname and the assigned ESB application name in the form

webservices.it.ucla.edu/<esb app name>/<service name>/<rest of context path>

Examples:
webservices.it.ucla.edu/ucpath/<service name>/…
webservices.it.ucla.edu/idm/<service name>/…
webservices.it.ucla.edu/cb/<service name>/…
webservices.it.ucla.edu/pams/<service name>/…
webservices.it.ucla.edu/iwe/<service name>/…
webservices.it.ucla.edu/opus/<service name>/…
webservices.it.ucla.edu/iamucla/<service name>/…

Note- For the non-production Test and QA environments, webservicesqa.it.ucla.edu and webservicesqa.it.ucla.edu will be registered.

For batch service interfaces, the files are sent to fx.it.ucla.edu over sftp. The same address is used for Test and QA environment testing.
This facilitates address standardization of services hosted in the bus, a framework to procure server certificates for provider applications and sign client certificates for consumer applications by the private CA, besides the branding and differentiation for the platform.

Note- For applications with existing services in production and moving to ESB, the addressing requirements can be relaxed until all their clients have changed to the new addressing scheme for both real-time and batch services, but are encouraged to do so at the earliest.

4.1.4 Security Requirements

Application security requirements around the following need to be determined.

- Transport or message level security for services
- Batch service FTP accounts or ssh keys
- Provider application identities (server SSL certificate)
- Consumer application or client identities (user/passwords, client SSL certificates)

Provider application’s PKI (private key and certificate) are provisioned and signed as part of the onboarding process. Client certificates for mutual auth ssl need to be signed by the ESB’s private CA. A java keystore file named <ESB app name>.jks (for e.g. ucpath.jks) with the application’s and its client certificates is maintained in the application’s containers’ etc folder.

4.1.5 Capacity Requirements

Based on the volume, throughput (or concurrency) and frequency of an application’s services usage, the following need to be determined.

- Number of containers
- Memory, disk, CPU, and network resourcing
- Fault tolerance and clustering topology

One or more containers will be provisioned for each application depending on its needs. Note - Though it is possible for applications to share containers because of the underlying OSGI framework, each container will be provisioned for one application only. This ensures application isolation at the process and file system (logs, properties and configurations) and for management operations.

4.1.6 Message Persistence

Applications that need brokers for messaging use cases have the choice of file-system or database persistence.

For file-system persistence, the ESB infrastructure uses a shared network file system (NFSv4), supported by IS. For database persistence, MS SQL Server dbms is used, supported by IMS. Both these dependencies are guaranteed to be secure, clustered, highly available, optimized for performance and governed by ITServices support processes.

The general guidelines for applications to choose one mechanism over the other are summarized in the following table.
<table>
<thead>
<tr>
<th>QoS</th>
<th>DBMS</th>
<th>NFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>DBMS better</td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td></td>
<td>Preferable to scale to an order of hundreds of messages per second or higher</td>
</tr>
<tr>
<td>Slow processing like long lived transactions</td>
<td>DBMS better</td>
<td></td>
</tr>
<tr>
<td>Fast processing for short lived transactions</td>
<td></td>
<td>NFS better</td>
</tr>
<tr>
<td>Cost to application</td>
<td>DBMS higher</td>
<td>NFS cheaper</td>
</tr>
<tr>
<td>Ease of configuration (for admins)</td>
<td></td>
<td>NFS better</td>
</tr>
</tbody>
</table>
4.2 Application Best Practices

The ESB platform provides lots of options or choices to design an integration solution. However the recommendation is that teams exercise best practices such as the following.

a) Avoid resource heavy application business logic or state in the ESB.

Fig – Application component deployment

The ESB platform is an integration and messaging platform and not an application server (though it can be used as a full-fledged J2EE application server).

Resource heavy application business logic, transaction & data processing belong in separate application servers (either dedicated to the application or a shared infrastructure) sized appropriately for its capacity and resources.

Access of a transactional database, directory server (e.g. LDAP) or mainframe and processing of data around those accesses would generally be categorized as core application business logic and state. Routing, protocol and technology mediation, transformation, messaging and orchestration logic would be
considered processing appropriate in the ESB. Applications are recommended to decouple and layer their
architectural components.

Applications could still architect to perform heavy processing in the ESB and opt to be charged based on
the capacity and computing resources used.

b) Discuss an expiry date for transformations.

Transformations are needed when consumer and provider applications don’t support the agreed message
format. This is usually because either or both are existing or legacy applications, and there is a time or
resource constraint to change either application to the agreed format. Especially in the context of real-
time interfaces but applicable to batch services as well, transformations (which can get complex) add
processing, error handling and performance overheads. Effort should be made to change the consumer
or provider application over time if not possible sooner or get a commitment for the change or even an
expiry date for the transformations.

c) Consider asynchronous, eventing or messaging for long-lived transactions.

For short-lived transaction or request processing (also related to the request size), web services are ideal.
For long-lived transactions (also large size requests such as batch processing) messaging is
recommended. Using messaging for short-lived transactions though not discouraged, adds complexity,
error handling and performance overheads.

d) Be reminded that Security & Performance are inversely proportional

A balance is recommended when choosing security policies (transport or message level, messaging) so
as to not add a performance overhead while addressing the SLA and security use cases of the
application.

e) Messaging queues belong to the provider application

The application processing the messages from a queue, the provider application or Message Listener,
owns the queue. The queue exists in a message broker provisioned for the application and adheres to the
policies of that application. Applications sending messages to the queue (the consumer application or
Message Sender) need to provide authentication credentials of the provider application queue.

f) Application should not open their own ports

Since the ESB is a shared platform hosting multiple applications, opening ports cannot be guaranteed to
work, as they might conflict with another application. The platform will provide through reserved
container system properties, ports that the application can read and open. This way the application can
be programmed to property keys, leaving the value to the infrastructure to fill in or change as the need
arises.

Note- The platform will expose other properties as well using this mechanism for applications. The list of
reserved property keys with descriptions is to be published separately.

g) Securing Personal Information (PI) data

Message level security is mandatory as per UCLA Policy 404 to address this concern. *(Read section
5.1.3)*. The platform by itself doesn’t deal with any PI data. However applications deployed in the platform
may. The platform does not automatically enforce this security for the application for every single
message as it will be a performance hog.
For e.g., a message with PI data is sent from one application to another, routed and/or transformed through one or more intermediaries in ESB. The PI data part of the message is secured (encrypted and signed). Unless the routing/ transformation application in ESB needs to handle the PI data as part of the business process, the secured data of the message should ideally be left untouched (without decrypting and encrypting again). The symmetric or asymmetric keys for the security should be exchanged between the end applications exchanging the message (and not with the ESB application).

The above consideration is optimal when the number of end applications is just two or a small set (say five). When the number of end applications is more than that, for e.g. when the ESB application is brokering communications between one application on one side and a number of applications on the other, the case can be made to decrypt and encrypt in the ESB application, to avoid the inconvenience of key exchange and management. In this case, utmost care needs to be taken, if the message is persisted in a file system (e.g. logs) or database (message queue persistency layer) after decryption, and on errors.

h) Use chunking and streaming when processing large sized data

Especially when processing large batch files in ESB, chunking and streaming techniques should be employed so as to not load the whole data into memory. Loading large sized data into memory would require the container to be configured with high memory configurations.
4.3 Deployment from ITServices ALM process

The above diagram displays the process of how applications are deployed into the ESB platform. For integration applications developed at ITServices, the deployment principle is to download build artifacts (internally built or external library dependencies) from the enterprise archiva as the system of record using the default maven download capability of the bus software stack.

(Note - For other campus department applications, the process is TBD).

Build artifacts are identified typically by their maven coordinates (groupid/artifactid/versionid). The ALM process defines how artifact coordinates are to be determined and evolved through the cycle. Build and Release management are outside the scope of the ESB platform and hence this document.
4.4 Application Configuration

Applications are configured in the ESB as **Fuse Fabric Profiles**. A profile is the basic unit of deployment in a fabric. You can deploy one or more profiles to a container and the content of those deployed profiles determines exactly what is installed in the container.

A profile encapsulates the following kinds of information:

- The URL locations of features repositories.
- A list of features to install.
- A list of bundles to install (or more generally any suitable JAR package—including OSGi bundles and WAR files).
- A collection of configuration settings for the OSGi Config Admin service. (see OSGI Configuration Admin Service in the References Appendix).
- Java system properties that affect the Apache Karaf container (analogous to editing etc/config.properties).
- Java system properties that affect installed bundles (analogous to editing etc/system.properties).

A profile can have one or more parent profiles from where it inherits configurations. A profile is a versioned object specified with a version number. Profiles are exported from one environment and imported into another during the application development lifecycle.

Each application has a base or parent profile named after the application (i.e. ucpath) with its core configuration, and a child profile for each environment (ucpath-dev, ucpath-test, ucpath-qa, and ucpath-prod) containing environment specific configurations such as properties.

**Note** - In the **standalone** ESB deployment mode, there are no central management servers, and an application’s osgi artifacts are downloaded to the container directly from the maven repository, by authenticating and executing a command in the container’s karaf console or operation in the web console.

![Fig – Fuse Fabric Profile in Management Console](image-url)
The following table enlists how different profile components are identified and specified.

<table>
<thead>
<tr>
<th>Profile Component</th>
<th>Identification / Specification</th>
</tr>
</thead>
</table>
| OSGI feature repositories | Identified in xml format. Maven coordinates  
mvn:groupid/artifactid/versionid/xml/features              |
| OSGI features          | Feature name from a feature repository                                                        |
| OSGI bundles           | Maven coordinates  
mvn:groupid/artifactid/versionid                                                                                  |
| OSGI properties        | Identified in config or property files in a container’s etc folder named with a persistent id (pid).  
Specified as name = value pairs.                        |
| System properties      | Name = value pairs                                                                               |

Maven repositories are configured in the **default profile** which is the root of the profile hierarchy. The **org.fusesource.fabric.agent.properties pid** file’s, **org.ops4j.pax.url.mvn.repositories** property is set with a comma delimited list of http urls, such as the following

```text
org.ops4j.pax.url.mvn.repositories=https://its-alm-esb-dm@ucla.edu:<password>:@archiva.it.ucla.edu/archiva/repository/itservices/@snapshots@id=its.repo
```

In Prod & QA environment, the role of application configuration is performed by ESB administrators (IMS team). Application profile/s are likely imported from a Test environment where they have gone through the quality cycle.

### 4.5 Application Deployment

Application deployment occurs when a **profile** is associated to a **container**. Deployment is an authenticated operation on the ESB, performed by an admin operator in the management console or by executing a unix script, providing ESB credentials. The ESB in turn uses the maven repository configuration specified with authentication credentials to access and download artifacts and its dependencies. In Prod & QA environment, the role of application deployment, is performed by IS administrators.
Security in the platform is addressed in two contexts, applications deployed in the platform, and the base platform security primarily for administration and management operations.

5.1 Application Security

5.1.1 Service Management

The Service Manager application component (for short svcmanager) is the centerpiece for the security and management of applications’ real-time and batch communications. The objectives of the application are as follows:

a) Separate the security concern from applications

Applications can focus on integration logic, and identify the security options that they want enforced and agreed with their consumer or client applications, and the platform enforces it.

b) Support standards based security mechanisms

For the first release of svcmanager, SSL enforcement is provided as the transport level security mechanism for real-time services. Real-time encrypted ssl traffic passes through the firewall and load balancer and terminates or offloads at svcmanager. Svcmanger then routes requests based on the esb
app name in the context path, webservices.it.ucla.edu/<esb app name>/<service name>/<rest of context path>.

Over future releases of the component, features such as message level security based on WS* and others, authentication and authorization (tied to identity management) will be added and enhanced. Further the platform evaluates technology trends and emerging standards and based on requirements, incorporates and makes available to applications. Applications hence don’t need to address the low-level details of integrating a new security protocol or mechanism.

c) Facilitate management and monitoring for applications

With svcmanager mediating all application services in the platform, real-time and batch, management (e.g. security configurations) and monitoring (e.g. audits) features can be provided and enhanced.

In the first release, svcmanager can mediate batch service communications. When a file arrives on the ftp server intended for processing by an application in the esb, a “batch arrived event” is generated to a svcmanager queue in the esb. Certain attributes from the file transfer such as sender account, receiver account, filename, etc. are captured. These are fed into the monitoring component. Svcmanager then routes the event based on the captured attributes to the specific application. The application can keep listening for the event from svcmanager, to trigger a request to the ftp server to pick up the file and start processing it. Svcmanager does not process the batch file itself but only the event.

Besides the benefit of the monitoring capability with this approach, applications don’t need to continuously poll the ftp server for their incoming files thus optimizing ftp server traffic and network resource bandwidth.

As a general design principle svcmanager tries to process mediations using the message routing and filtering EIPs as opposed to content based routing and filtering. This implies that routing or filtering decisions for monitoring try to rely on message headers instead of parsing and processing data elements from message payloads, which can add a performance overhead to interfaces.

d) Eventually support SOA governance

SOA governance generally refers to the ability to categorize services into groups, even hierarchical groups, such as business units (teams, departments, etc.), define common sets of policies (security, management & monitoring) for a group, save the policies typically in a registry (e.g. uddi) from where they can be discovered, enforce and mediate the policies between consumer and provider applications, be able to extract runtime data and perform statistical analysis on it for alerting, SLA enforcement and reporting purposes, at any group level. The effective policy of a service interface or operation is a logical union of policies defined in its group hierarchy.

SOA governance improves operational efficiencies when services deployed in the platform are a large set, as the platform is continuously adopted by applications and departments. Configurations or policies are not laboriously defined for each service or interface but common policies are defined for groups at appropriate levels. Without this capability, managing a very large set of campus services is not optimal.

The svcmanager component becomes the logical integration point for SOA governance enhancements and solutions for the esb platform.

As can be inferred, that security enforcement is the necessary first step to make available a whole gamut of management and governance capabilities for applications and a unified SOA architecture roadmap through the platform. The component name is in keeping with this broad perspective.
5.1.2 IP Filtering

IP filtering is a feature that can be enforced for both real-time and batch communications. IP filtering is enforced at the firewall. By default, inbound access from the “world” to the ESB platform is disabled. Clients of provider applications have to be added during application onboarding and via service requests thereafter, to the platform’s IP address whitelist.

5.1.3 Transport and Message level security

Transport level relates to in-flight or in-transit security, provided by a transport protocol such as http or ftp. The security terminates at the end of the transit. There is no choice to secure some parts of the data and leave other parts insecure, over a communication channel. Also data that is flowing through multiple intermediaries like in the case of ESB, each point to point communication is secured separately. It involves processing transport headers. It adheres to University of California Policy IS-3 Electronic Information Security. SSL/TLS & SSH are popular mechanisms that provide transport level security.

(Note- In the OSI model, SSL/TLS is enforced between layers 4 & 5. That’s how several application layer (5 & above) protocols are able to leverage ssl, be it http, ftp, ldap, jms, etc. Software & application frameworks, environments, stacks (like JRE of which JSSE is part of) abstract all this.)

Message level relates to security enforced outside the transport channel generally in an application layer. Here the whole data message or just sensitive parts of the message can be secured, there is a choice, considering sensitivity and performance. Also at message-level, there is a choice of where the message is decrypted, it does not have to be enforced at each intermediary. It involves processing message headers, such as soap headers. It satisfies the security in storage requirement part of UCLA Policy 404 Protection of Electronically Stored Personal Information. Security in storage doesn’t imply to only data in an application database, but also log files, message queues, etc. along the way (outside the scope of communication channels).

5.1.4 Real-time service security

Real-time service security involves the security of web services (SOAP & RESTful), exposed by provider applications in the platform, which basically communicate messages in xml format over HTTP. The general security concerns of privacy, authenticity, authorization, non-repudiation and audit will (over phases) be addressed in a unified approach through the svcmanager component. Until then applications can build their own solutions using the base platform frameworks such as CXF and others for transport and message level security.

5.1.4.1 Ports

The svcmanager component exposes three HTTP ports to service real-time traffic, 80 for plain http, 443 for basic https, and 444 for 2 way https. Thus the addressing scheme discussed in section 4.1.3 results in the following addresses

Plain http - http://webservices.it.ucla.edu/<esb app name>/<service name>/<rest of context path>
Basic https - https://webservices.it.ucla.edu/<esb app name>/<service name>/<rest of context path>
2 way https - https://webservices.it.ucla.edu:444/<esb app name>/<service name>/<rest of context path>

The load balancer passes traffic for the host name on the three ports to the svcmanager component on the same ports.
5.1.4.2 PKI

Private-public key pairs (PKI) will need to be provisioned for the platform and applications, for enforcing transport and message level security. Application level pkis will be provisioned and signed during the onboarding process. All server certificates will be signed by the UCLA contracted InCommon certificate authority (CA). Client certificates for services requiring client certificate enforcement, will be signed by the ESB's private CA.

Clients or consumers can choose to either

a) Create their own pki and send over a certificate signing request (csr), to be signed by the esb platform. This is the recommended and more secure option as the private key of the pki is always only with the client.

b) Have the esb platform provision the pki and sign it.

Open question – How do we define application pki DNs (distinguished names)?

5.1.4.3 Privacy

The privacy or encryption concern is supported by the following options

a) HTTPs – transport level

b) WS-Security XML Encryption – message level

Note - HTTPs support will be available in the first release of svcmanager component. WS-Security XML Encryption support will be available in a later release.

5.1.4.4 Non-repudiation

The non-repudiation concern is supported by the following options

a) WS-Security XML Signature – message level

Note – WS-Security XML Signature support will not be available in the first release of svcmanager.

5.1.4.5 Authentication

The authenticity concern is supported by the following security token options

a) HTTP Basic Authentication – transport level

b) WS-Security UsernameToken - message level

The following advanced options can be considered based on application needs

c) WS-Security BinarySecurityToken

- X.509 certificate – for certificate based authentication
- Kerberos ticket
- SAML token

d) HTTPs with client authentication – transport level

e) WS-Trust – Secure Token Service

f) OAuth

The result of authentication is an identity, which is discoverable in an identity management system. Through the JAAS framework of the esb platform, authentication check against an identity management system is made possible.
Open question: Integration with UCLA’s identity management system is to be explored. The need is to support application and end-user identities. Initially a private LDAP identity store is leveraged to maintain user/passwords.

Note – HTTPs with client authentication is to be available in the first release of svcmanager.

5.1.4.6 Authorization
After an identity is available through authentication, authorization checks whether the identity has the necessary privilege or permission to access the requested resource or interface. The JAAS framework of the esb platform is leveraged for authorization check.

5.1.5 Batch service security
This relates to the security of batch services exposed by provider applications in the esb platform. Batch service communication happens when a client application sends a file destined for a provider application, to the FX server over ftp. FX server can be configured to send a message event to the svcmanager component in the esb. Svcmanager in turn notifies the provider application, which then collects its file from FX server, and starts processing it.

Thus batch service security addresses, the file transfer between the application and FX server, and the message event between FX server, ESB and the application. The security around the message event is addressed by the Messaging or MQ security section later.

5.1.5.1 Privacy
The privacy concern is addressed by using secure ftp or SFTP in the communication between the application and FX server.

5.1.5.2 Authentication
Each application communicating with FX needs to have an account in the server. Applications requiring password-less transfers, can generate a ssh key (using a tool like OpenSSH) with at least a 2048 bit size. Either password or key or even both can be used in the communication. Authentication is enforced in FX server. The account represents an application identity.

Note – It is to be explored how to address the application account identity in a unified identity management solution approach.

5.1.6 Messaging or MQ security
Messaging can be used both in real-time or batch service processing, to breakup long running transactions, by introducing asynchronous behavior, hence improving performance. Security involves policies at 2 levels, at the broker level, and at the destination level (queue or topic).

Every broker belongs to an application. An application uses broker credentials when communicating with destinations in its broker. However if a destination is exposed to other applications, destination credentials need to be used to communicate with it.

SSL can be enabled for MQ communication. Messaging also has firewall constraints and not allowed across the firewall.

Note – More details in a later revision.
5.2 Platform Security

This involves the security of administrative operations when using Web, SSH & JMX consoles, and other interactions in the underlying esb platform.

When administrators interact with the consoles, the JAAS framework of the platform is leveraged to integrate with an identity management system for end users. Web consoles for management and monitoring are served over https.

Note – In the first phase, a private LDAP is used as an identity store to maintain user credentials. The interaction between the platform and the LDAP instance itself is over SSL.

The interactions between the management servers and the distributed containers are over ssh and jmx. Operations with root containers are executed over ssh using the local VM account. Operations with child containers are executed over jmx using user accounts from the JAAS integrated system.

The interactions between the platform and DBMSs for message persistence and monitoring, and NFS for file system based message persistence, rely on the native security provided by those systems.
6. Monitoring

6.1 Monitoring Solution

A comprehensive robust monitoring solution is a key need for a platform such as the ESB. The solution needs to address monitoring of the platform as well as applications deployed in it. The following resources especially have to be supported - Apache ActiveMQ, Apache Camel, Apache CXF, JBoss Fuse & JBoss Fuse Fabric.

Evaluations of the following two industry enterprise monitoring solutions are pending. Details will be available after the evaluations in a later revision.

6.1.1 JON

Redhat’s JBoss Operations Network (JON) is the vendor recommended solution with tight integration with the JBoss Fuse product suite. This includes support and feature upgrades for the tool. However Fuse and JON have separate licensing, costing, terms and conditions.
Fig – Container status

Fig – JON Dashboard
6.1.2 Nagios

Nagios / Icinga is proposed to be used by ITServices IS for network/systems monitoring at ITServices. Some generic capabilities are available using JMX, and for Apache Camel. It would likely require customizations to integrate fully with the Fuse product suite for a quality solution that will need development cycles.

6.2 Log Monitoring

Splunk log monitoring application supported by ITServices IS enables teams developing and supporting their integrations in the esb platform debug the behavior of their applications hence monitoring the health.
4 ESB environments (Dev, Test, QA and Prod) are to be configured for Splunk. QA and Prod environments are to be configured to ITServices’ Prod Splunk instance (with 10GB daily limit license). Dev & Test environments are to be configured to a separate dedicated instance for the ESB platform supported by IS with a 1GB daily limit license. This is to support active ESB application development cycles while not affecting Production Splunk monitoring operations.

6.3 Fuse Management Console
JBoss Fuse Management Console provides basic monitoring features in addition to its primary management features. Container status, JVM details, Message Broker details including for a queue or topic, Camel route details are some of the runtime information available.
Fig – Container status

Fig – JVM details
Fig – MQ Broker details

Fig – MQ Queue or Topic details
Fig – Camel Route details
## 7. Appendix A- Abbreviations & Acronyms

<table>
<thead>
<tr>
<th>Topic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB</td>
<td>Enterprise Service Bus</td>
</tr>
<tr>
<td>JRE</td>
<td>Java Runtime Environment</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>JMX</td>
<td>Java Management Extensions</td>
</tr>
<tr>
<td>JAAS</td>
<td>Java Authentication &amp; Authorization Service</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocol</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>IMS</td>
<td>ITServices Information Management Services</td>
</tr>
<tr>
<td>IS</td>
<td>ITServices Infrastructure Services</td>
</tr>
<tr>
<td>UCPATH</td>
<td>UC Payroll, Academic personnel, Timekeeping &amp; HR project</td>
</tr>
<tr>
<td>BruinBill</td>
<td>CB or Consolidated Billing project</td>
</tr>
<tr>
<td>PAMS</td>
<td>Post Audit Management System project</td>
</tr>
<tr>
<td>IWE</td>
<td>Integrated Web Experience project</td>
</tr>
<tr>
<td>OPUS</td>
<td>Faculty Information System project</td>
</tr>
<tr>
<td>IAMUCLA</td>
<td>Identity Management program</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
</tbody>
</table>
## 8. Appendix B- References

<table>
<thead>
<tr>
<th>Topic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSGI</td>
<td><a href="http://www.osgi.org">http://www.osgi.org</a></td>
</tr>
<tr>
<td>OSGI Configuration Admin Service</td>
<td><a href="http://www.osgi.org/javadoc/r4v42/org.osgi/service/cm/ConfigurationAdmin.html">http://www.osgi.org/javadoc/r4v42/org.osgi/service/cm/ConfigurationAdmin.html</a></td>
</tr>
<tr>
<td>UC Policy IS-3</td>
<td><a href="http://www.ucop.edu/ucophome/policies/bfb/is3.pdf">http://www.ucop.edu/ucophome/policies/bfb/is3.pdf</a></td>
</tr>
</tbody>
</table>
9. Appendix D- ESB at ITServices

Fig - ITServices Centric View
10. Appendix E- Open Items

The following are the open Architectural items at the time of this revision.

10.1.1 Disaster Recovery

The ESB platform is going to be a critical infrastructure service to the campus, especially as we host more applications and services over time. **Should it be considered a tier-1 service?** If it is a tier-1 service, a DR plan has to be devised, and a cost-benefit analysis needs to be done. **Do we need to go as far as building an ESB infrastructure in another data center?** If the server(s) hosting the infrastructure experience catastrophic failure, tenants suffer total outage, **should we rely on the host VMware infrastructure providing DR for this type of event?**

10.1.2 Campus development environment

Most UCLA campus IT development happens on non-j2ee platforms, whereas the ESB platform development is on the j2ee platform. Application services hosted in the platform are interoperable by being standards based, however development is tightly tied to the j2ee platform. **How do we bridge this gap?** This isn’t an issue for ITServices projects and applications.

Possibly integration components for non ITServices non-j2ee projects, if architected right and decoupled from other application components, need to be developed, deployed and supported by the ESB team at ITServices.

Will development teams outside ITServices work on ESB SDLC environments? If yes, will they need access in ITServices ALM process (i.e. SVN, Teamcity, Archiva, Jira)? Will they need access to Splunk to monitor logs? There might be other ITServices processes that might need to be exposed outside.

10.1.3 Identity management system

ESB platform integration with UCLA’s identity management system (IAMUCLA) for application and end-user identities needs to be addressed.

10.1.4 ALM process

Since the ALM process is relatively new, we need to ensure that application lifecycle use cases related to the ESB platform are efficiently supported by that process. ESB platform is a **J2EE server**, different from traditional J2EE servers, in that it supports **Maven based deployments**. The default deployment mechanism of the platform is to download and install artifacts from a Maven repository.

Two concerns in the current ALM process that need to be addressed **generically** affect ESB deployments.

1. Support for project development milestones – Currently ALM process supports only a direct production milestone. Any software or IT project plan and development is typically divided into milestones with deliverables. The milestone deliverables are generally available during the project lifecycle (the production or final release milestone lives much longer beyond that) to flexibly at any time in the QA cycle, deploy into any of multiple Dev or Test environments.

(Note – As of this revision there is consensus in ALM workgroup for this need. Next we need to see how it is implemented in the process).
2. J2EE platform supports different artifact types (i.e. jar, war, ear, xml, etc.). Though different artifact types may need to be handled or processed differently in ALM process, the benefits or features of the process should be available for all types.

10.1.5 **JON Monitoring solution DBMS requirement**

The JON monitoring product stores all its data in a DBMS. Only 2 DBMSs are supported, **Oracle** and **Postgres**. Although **MSSQLServer** is the preferred and best supported DBMS at ITServices, between the two, Oracle is preferred.

The existing Oracle infrastructure doesn’t support high availability, which means DBMS maintenance (for e.g. patches, upgrades) requires downtime. This doesn’t affect traffic through the ESB, however monitoring of the traffic is.

JON support has informed that the product architecture roadmap doesn’t include DBMS, but rather a big data file system based storage such as **Apache Cassandra**. They have begun the phased development of their new architecture which will be available through upcoming releases. The product will have no DBMS dependency in **2015** as per current estimates.

In this context, investment in a highly available Oracle infrastructure may not be warranted.

10.1.6 **Global File System (GFS)**

GFS provides better scalability and performance compared to NFS as a distributed file system for message persistence and data sharing. Both NFS and GFS were evaluated for message persistence functionality. Though GFS is freely licensed, there are additional costs with RHEL connectivity.

Wouldn’t the additional license costs be warranted for a scalable high volume & throughput ESB platform?