Document Title:	HNG-X Architecture - Branch Access Layer
Document Type:	Architecture (ARC)
Release:	Not Applicable
Abstract:	Describes the software infrastructure and architecture that will be the basis for implementing the business requirements of HNG-X within the realm of the Branch Access Layer.
Document Status:	APPROVED This document contains sections that have been identified to POL as comprising evidence to support the assessment of named Acceptance Criteria by Document Review. These sections must not be changed without authority from the FS Acceptance Manager.
Author & Dept:	Andy Thomas
Internal Distribution:	
External Distribution:	

Approval Authorities:

Name	Role	Signature	Date
Roger Wright	CTO		
Alan D'Alvarez	Programme Manager		

Note: See Post Office Account HNG-X Reviewers/Approvers Role Matrix (PGM/DCM/ION/0001) for guidance.

0 Document Control

0.1 Table of Contents

0.2 Figures

0.3 Document History

Version No.	Date	Summary of Changes and Reason for Issue	Associated Change - CP/PEAK/PPRR Reference
0.1	27-OCT-2006	Draft	
0.2	07-NOV-2006	Draft for review	
0.3	26-NOV-2006	Draft for review	
1.0	16-FEB-2007	Version for approval.	
1.1	24-JAN-2008	Many updates based on architectural changes	
1.2	22-OCT-2008	Updated with Architectural Review comments	
1.3	01-DEC-2008	Updated with further comment responses.	
1.4	09-Dec-2008	Insertion of new Section 0.5 containing the table of cross references for Acceptance by Document Review.	
1.5	12-Jan-2009	Addition of new entry into Acceptance by Document Review Table.	
1.6	10-Jul-2009	Revised circulation list, draft for review	
2.0	16-Oct-2009	Version for approval	

0.4 Review Details

This document is subject to Group Review. Mandatory Reviewers should consult the Author to determine the details of the associated Group Review before submitting a formal comment sheet.

Review Comments by :	
Review Comments to :	
Mandatory Review	
Role	**Name**
Solution Design	Adam Cousins
Infrastructure Design	Pat Lywood
HNG-X Service Transition	Graham Welsh
Security Architect	Jim Sweeting
Information Governance	Brian Pinder
System Qualities Architecture	Dave Chapman

POL00140245

Optional Review	
Role	**Name**
Security & Risk Team	GRO
Architect	Jason Clark
Architect - Counter	Ben Holland
Architect - Bal	Andy Thomas
Test Design	George Zolkiewka
Service Network	Ian Mills
Service Support	Kirsty Gallacher
System Test	John Rogers
SV&I Manager	Sheila Bamber
Tester	Hamish Munro
RV Manager	James Brett (POL, JTT)
VI & TE Manager	Mark Ascott
Testing	Stephen Gilbert
Head of Service Management	Gaetan van Achte
SSC	Mik Peach
Business Continuity	Adam Parker
Head of Service Change and Transition	Graham Welsh
Data Centre Migration	Geoff Butts
Data Centre Migration	Peter Okely
Programme Manager	Alan D'Alvarez
Integrity Testing	Alan Child
Integrity Testing	Michael Welch
Security	Peter Sewell
Core Services	Ed Ashford
Core Services	Andrew Gibson
CTO	Roger Wright
Documentation	Trish Morris
Issued for Information – Please restrict this distribution list to a minimum	
Position/Role	**Name**
Acceptance Manager	David Cooke

0.5 Acceptance by Document Review

The sections in this document that have been identified to POL as comprising evidence to support Acceptance by Document review (DR) are listed below for the relevant Requirements:

POL NFR DR Acceptance Ref	Internal FS POL NFR Reference	Document Section Number	Document Section Heading
SEC-3199	SEC-3152	2.5.5	Authentication/authorisation architecture
SEC-3226	SEC-3226	2.2.5.1	Use of Statements and PreparedStatements

0.6 Associated Documents (Internal & External)

Reference	Version	Date	Title	Source
PGM/DCM/TEM/0001 (DO NOT REMOVE)			Fujitsu Services Post Office Account HNG-X Document Template	Dimensions

ARC/APP/ARC/0001			HNG-X Reference Data Architecture	Dimensions
ARC/APP/ARC/0008			HNG-X Branch Database Architecture	Dimensions
ARC/APP/ARC/0009			HNG-X Counter Business Application Architecture	Dimensions
ARC/APP/RTM/0004			Requirements Traceability Matrix for Batch Applications	Dimensions
ARC/GEN/PRP/0001			HNG-X logon and authorization options paper	Dimensions
ARC/GEN/REP/0001			HNG-X Glossary	Dimensions
ARC/NET/ARC/0001			HNG-X Network Architecture	Dimensions
ARC/PER/ARC/0001			HNG-X System Qualities Architecture	Dimensions
ARC/PPS/ARC/0001			HNG-X Platforms And Storage Architecture	Dimensions
ARC/SEC/ARC/0003			HNG-X Technical Security Architecture	Dimensions
ARC/SOL/ARC/0001			HNG-X Solution Architecture	Dimensions
ARC/SOL/ARC/0005			CTO Design	Dimensions
ARC/SYM/ARC/0001			HNG-X Systems and Estate Management Architecture	Dimensions
DES/APP/HLD/0015			HNG-X BMX Monitor High Level Design	Dimensions
DES/APP/HLD/0050			HNG-X Online Service Routing High Level Design	Dimensions
DES/APP/HLD/0094			HNG-X End-To-End Key Usage Overview HLD	Dimensions
DES/APP/IFS/0012			BAL Interface Specification	Dimensions
TST/GEN/STG/0001			HNG-X Testing Strategy	Dimensions

Unless a specific version is referred to above, reference should be made to the current approved versions of the documents.

0.7 Abbreviations

See also HNG-X Glossary (ARC/GEN/REP/0001).

Abbreviation	Definition
API	Application Programming Interface
BSM	Branch Session Manager
DAO	Data Access Objects
DOM	Document Object Model
DTO	Data Transfer Object
DVLA	Driver and Vehicle Licensing Agency
EJB	Enterprise Java Bean

ESB	Enterprise Service Bus. An architecture that reduces the amount of point-to-point connections between servers and provides a 'bus' type topology.
HTTP	Hypertext Transport Protocol
IoC	Inversion of Control
IDS	Intruder Detection System
IPS	Intrusion Prevention System
J2EE	Java 2 Platform Enterprise Edition
JAXB	Java Architecture for XML Binding
JAX-RPC	Java API for XML-based RPC
JDBC	Java DataBase Connectivity framework. The standard method for Java applications to access relational databases.
Jibx	Framework for XML data binding to Java
JMS	Java Message Service
JMX	Java Management eXtensions
Local Data Access Services	Services that are accessed through the Interstage platform, and primarily interact with the Branch Database
NIO	Non-Blocking I/O
OCI	Oracle Call Interface. In this context of this document this refers to a pure Java Oracle database driver that does not require any native software components.
OSR	Online Service Router – custom server component for network routing of online services
SAX	Simple API for XML
SOA	Service-oriented architecture
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TPS	Transactions Per Second
XML	Extensible Mark-up Language
XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language Transformations

0.8 Glossary

See also HNG-X Glossary (ARC/GEN/REP/0001).

Term	Definition
Refactor	A commonly used word in software development for the activity of re-organising and re-structuring code for better structure and removing duplicate functionality.
Assertable	"capable of being affirmed or asserted" – in software terms software or a software components that allow you to assert a success with a guaranteed binary "true" or "false" result.

0.9 Changes Expected

Changes

0.10 Accuracy

Fujitsu Services endeavours to ensure that the information contained in this document is correct but, whilst every effort is made to ensure the accuracy of such information, it accepts no liability for any loss (however caused) sustained as a result of any error or omission in the same.

0.11 Copyright

1 Scope

1.1 Intended audience

The intended audience for this document is primarily architects, designers and developers within the HNG-X project. Other stakeholders who are sufficiently familiar with the basic technological concepts should also be able to find some useful information in this document, but they are not its primary target.

However, the document is not intended for a non-technical audience, as it is inherently technical in its nature.

1.2 Scope & limitations

This document is intended to describe the software infrastructure and architecture that will be the basis for implementing the functional requirements of HNG-X within the realm of the Branch Access Layer. The document does not intend to provide any details of the functional requirements themselves; instead the architecture should be seen as the infrastructure/framework within which the functional requirements will be implemented. The non-functional requirements described here include resilience, stability, security and authentication. How the functional requirements will be met will be detailed in more depth during the design phase.

In addition, the document addresses the following areas:

- Some of the major technological choices and their rationales (some other choices are detailed in HNG-X logon and authorization options paper (ARC/GEN/PRP/0001)).
- Service interface definition and best practices.
- Architectural description and high-level design for each of the major architectural topics identified.
- Practices for implementing individual services within the provided architecture/framework.
- Other areas of consideration, such as recommended testing practices, performance considerations etc.
- Impact of surrounding environment.

1.3 Background

Post Office Ltd operates in both the retail and financial services industries. The main channel to market for Post Office is a network of approximately 14,000 branches with volumes of up to 28 million customers per week. In addition, Post Office has been expanding the use of the Internet and Call Centres as part of a comprehensive multi-channel strategy.

The objective of the HNG-X programme is to develop a system with structural and operational characteristics that substantially reduce ongoing support and maintenance costs with respect to the current Horizon system.

The overall requirement is that the business capabilities offered by the current system (Horizon) are preserved in the new system (HNG-X). However, a limited number of business capabilities will be revised based on a joint optimisation of business requirements and system properties.

The analysis of the serviceability profile for Horizon has highlighted data management as one of the most significant drivers for cost. The storage of transactional data within counters causes the need for security mechanisms that impact both the structural complexity and the operational performance of the counter applications. In addition, the presence of sensitive data on the counter increases the time, complexity, and ultimately the cost of maintenance procedures.

The HNG-X solution is based on a set of business applications that support a centralised model for data storage. The counters retain operational data (e.g. Reference Data) and business logic, but transactional information is stored directly in the Data Centre.

The counter side of the new applications is based on Java technology. The counter hardware is reused from Horizon with the operating system being Windows NT.

The Branch Access Layer is also Java based, it forms the interface from the counter to the Data Centre services.

The Branch Access Layer (BAL) is the server side interface used by the Counter client application in the branch offices. The BAL has the responsibility for providing remote business services to the Counter, and handles all database access, routing to other remote back-end services, and server side audit and recovery writing.

1.4 Context

Three types of HNG-X architecture document exist:

- The **overall solution architecture** is the top-level description of the HNGX solution, identifying the individual topic architectures.
- A **topic architecture** is the first level of decomposition of the overall solution architecture. Topic architectures include Applications, Platforms, Networking, System Management, Security, Recovery and Resilience.

  If a topic architecture is complex, then a further decomposition into individual **component architectures** may be required. For example the System Management topic architecture could constitute Software Distribution, Monitoring, Remote Access and Diagnosis and Time Synchronisation component architectures. A component architecture identifies the scope for the associated High Level Design.

This document is the overall **topic architecture** for the **HNG-X Branch Access Layer**, and is derived from the outline architecture described in HNG-X Solution Architecture (ARC/SOL/ARC/0001), and will be used as input into the subsequent High Level Designs.

Some of the components listed in this document are also described in:

- HNG-X Technical Security Architecture (ARC/SEC/ARC/0003)
- HNG-X Systems and Estate Management Architecture (ARC/SYM/ARC/0001)
- HNG-X Platforms And Storage Architecture (ARC/PPS/ARC/0001)
- HNG-X Network Architecture (ARC/NET/ARC/0001)
- HNG-X Reference Data Architecture (ARC/APP/ARC/0001)

2 Architectural Description

2.1 BAL Environment

The BAL serves as the interface between the counter and other data centre systems. It communicates with the following servers:

- Authorisation Agents – this is to perform online financial transactions on behalf of a counter
- Web Service Agents – to allow a counter to communicate with other external systems.
- BRDB – the counter accesses the BRDB through the BAL servers
- Key Management Server – to retrieve any cryptographic keys or seeds that may be required for the BAL or the counter.
- Management Interface – A JMX based management server is used to control and monitor BAL servers.

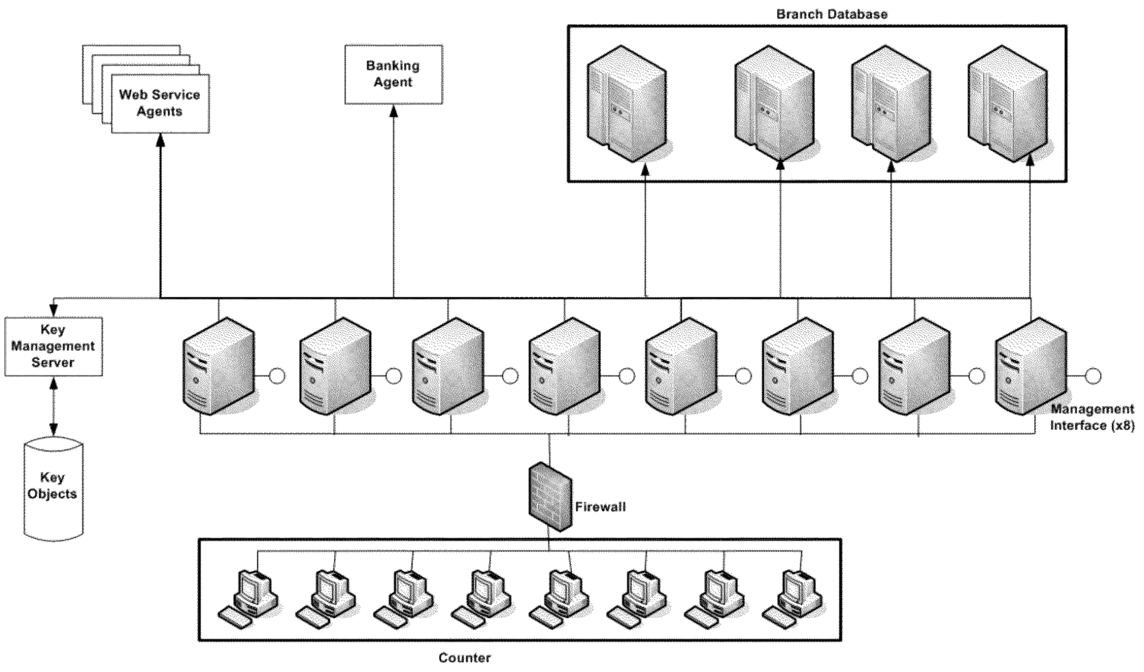

Figure 1 – BAL Environment

BAL servers are stateless, the number deployed can be easily altered so the eight servers shown in the diagram above should be used as a guide only, less or more servers could be deployed as is required.

2.2 Architectural/Technological choices

At the beginning of the architectural process, we faced a number of technological options which could be used. Some of these choices where clear-cut but needed to be made explicit. Others needed some clarification and validation.

This section details the choices of implementation technologies and their rationale. It is important to document these choices, in order for the project to be clear about them, and establish the technologies and practices to be used.

2.2.1 Use of Open Source

At the start of development the use of open source software was not allowed. This requirement leads us to develop some parts of the solution where otherwise we may have chosen to use open source. At later stages of the project this restriction was relaxed to some extent, allowing us to use some industry standard frameworks to replace and enhance what had already been developed. The transition from JDK 1.4 to 1.6 also allowed us to make use of more modern java API's and to remove some code that was not now required.

2.2.2 Minimal changes to Data Centre Infrastructure

There might have been a case for revamping the estate by creating a more unified integration backbone architecture using messaging and moving away from the current approach based on point-to-point integration with batch jobs and direct network calls to interrelated systems, which makes for a somewhat tightly coupled architecture. The architecture does open the way for the possible future implementation of an Enterprise Service Bus, something that should be considered as part of long term planning for the POL estate, but does not implement this in the HNG-X design.

2.2.3 SSL

The BAL does not accept SSL connections; the SSL is terminated at the ACE blade that sits in front of the BAL. This reduced the load on the BAL, but more importantly this maintains the system requirements around there being no encrypted traffic passing through firewalls. This will make the future implementation of IDS/IPS systems or XML firewalls possible.

More details are in the network description in Section 5.

2.2.4 Counter-BAL Communication Protocol

Communication between the counter and the BAL uses an XML over HTTP protocol. This is similar to a SOAP based approach but is simpler and more efficient.

The XML produced is valid XML that could reliably pass through network layers such as an XML firewall or similar device.

The XML is compressed using standard gzip compression to reduce the message size. Standard HTTP headers are use to indicate the compression, so this should not cause any network issues. As XML is highly verbose and contains much duplication of element names the compression ratio has been ascertained to be very high. The CPU overhead of decompression has been tested and determined to be low.

Details of the analysis used to decide on this protocol are provided in an appendix to this document.

In future the protocol could be changed if required, but this would impact the counter application, the BAL application and the Audit system. We have assumed, given the network topology and architecture that the most feasible and reliable mode of interaction between the counter and server is a traditional request/response synchronous mode of operation over HTTP. Other options that were explored, such as asynchronous HTTP callbacks to the client, raised too many unresolved questions and security issues to be pursued further.

It was also the recommendation of Fujitsu Software Japan to use a synchronous mode of interaction between counter and server.

2.2.5 Database access

This section outlines the approach and conventions used for database access; it defines methods of accessing, querying and writing to the database. Furthermore, database access will comply with any security requirements set out by the security architecture. .

2.2.5.1 Use of Statements and PreparedStatements

We will use *PreparedStatements* for database queries and updates from the Java side. PreparedStatements provide superior performance on queries that are performed several times during a system's uptime and are therefore preferred. They also provide protection against SQL injection attacks.

Furthermore, the J2EE community has a preference towards PreparedStatements, and their strength has been validated through numerous projects and their industry-wide preference over regular Statements.

2.2.5.2 JDBC driver

As the result of both research and experience of other implementations, the Oracle "thin" JDBC drivers will be used in preference to the OCI drivers.

2.2.5.3 Connection pooling and DataSources

Standard Java connection pooling mechanisms will be used to maintain long lived connections to the Oracle database. This minimises the impact on the database server as creating a new connection is an expensive operation.

More information around the database/RAC connection pooling and load balancing can be found in *section 2.5.6.2*

2.2.6 Service Oriented Architecture

The BAL layer design is based on a Service Oriented Architecture. Functionality is deployed in standalone service handlers. Service Handlers can perform a variety of functions, but they can be classified in to the following groups:

- Routing Services – these are services that pass the service message on to a back-end authorisation or web service (within the same data centre). BAL routing services may provide additional services for these messages such as logging, authorisation checking and message format translation.
- Data access services – services that provide read and/or write access to the BRDB. This includes services such as recovery management and audit logging.
- Business services – services that provide business functionality could possibly be deployed directly in the BAL. This is not a class of service that makes sense in most cases as there are other, more appropriate, locations that business functionality can be deployed. For example if a group of related business functions were required to be deployed it may be better to implement a stand-alone web service to isolate these functions in one or more dedicated servers. HNG-X also deploys business functionality directly in the counter application where appropriate.
- Login authentication and authorisation for counter messages.

Services are accessed by a URL reference that forms a part of the HTTP request on the BAL.

2.2.6.1 Service Versioning

When a service is altered in a way that makes it incompatible with its current implementation we need to support both versions of this service for a period of time. This is because counter updates cannot be deployed instantly.

Counters cannot normally be more than approximately 4 days out of date, if they get this far behind in incremental reference data updates they are forced to do a complete refresh before they can log on. An exception to this is service repair counters that may be some time out of date when they are first connected. They will immediately begin a download of up-to-date reference data, but this could take some time on a slow line. This is part of the reference data business stream and is transparent to the BAL (indeed it may actually be 10 days and can vary if necessary). Reference data is not distributed by the BAL but the BAL does participate in reference data updates by providing a polling service for the counter to determine if it should pull updated reference data. The reference data download does not occur through the BAL but through System Management.
This is totally separate from counter software management. For example, where we are piloting some new business software it could be several weeks before we distribute it to the whole estate. The way that we support multiple versions of a service is by deploying each of these versions in the BAL using a different service name. There will be a naming convention to identify versions of the same service.

2.2.7 Management and configuration

The cost of management and maintenance of the HNG-X application is an important factor for the decision to embark on this project.

For this reason, we will use a combination of two complementary approaches to achieve ease and lower cost of management:

- Use of JMX (Java Management eXtensions), a standard Java approach to management, monitoring and configuration of large-scale Java applications, supported by most application servers (directly or indirectly) and management tools.
- Use of an "Inversion of Control" (IoC) pattern for the development of the HNG-X server architecture and services. Inversion of Control is a pattern that moves configuration settings from Java code to configuration files. This lessens the amount of object creation code developers have to write so that the total code base may shrink by as much as 30%. IoC also fits in well with the approach JMX takes to configuration and management, making it a perfect fit for ease and efficiency of management.

2.2.7.1 JMX

Java Management eXtensions (JMX, http://java.sun.com/products/JavaManagement/), is the standard approach in Java for management, monitoring and configuration of larger scale applications and enterprise applications. JMX provides the following benefits to mention a few:

- JMX is a management framework that plugs into many different management agents/tools, making it usable from a wide variety of both commercial and free management tools.
- Unified API: JMX is a standard, and therefore can be expected to be widely understood by architects, developers and tool makers now and in the future.
- Scalable management architecture: JMX is appropriate from everything to small applications to very large and complex enterprise applications.
- Enables runtime control over configuration: with the help of JMX, a system can be reconfigured at runtime in most cases.

- Monitoring: JMX enables you to define Event Notifications, enabling the managed application to transparently notify a management agent/tool of certain critical and/or important events.

2.2.7.2 Inversion of Control Framework

The open source Spring framework is used for IOC. Spring is a well known standard framework that most Java developers are familiar with.

2.2.8 Authentication and authorisation

2.2.8.1 SRP Authentication

The SRP Protocol (RFC 2945) is used for login authentication. This protocol is a well documented and used as standard for remote authentication and therefore has had the benefit of a through analysis by security professionals. SRP has built in defences for various attacks such as man-in-the-middle attacks.

More details of the login process can be found in section 2.5.5.1.

2.2.8.2 Digital signatures

Digital signatures will be used because of two major factors:

- Authorisation: after initial logon, a user & counter can be identified through their signature rather than anything else.
- Tamper-proofing the audit trail: there is a requirement for the audit trail from HNG-X to be able to stand up in court, even if someone would be able to access and edit the audit trail. The only way of coming anywhere close to tamper-proofing the audit trail is through the use of digital signatures, since these are very hard to forge. To ensure the long-term ability to validate signatures, session public keys will be stored with the corresponding audit records.

Within the context of a user session the counter will sign every message sent to the BAL. The BAL will check the signature on every message. The counter uses its Counter Message Signing Private Key to sign messages.

The BAL signs login messages from the counter and stores the signed version along side a copy of the BAL Message Signing Certificate. The BAL signs the login message to provide proof that the counter login is legitimate, because this message includes the Counter Message Signing Public key it provides a chain of authentication to validate each counter message for that login session.

2.2.9 Encryption

The BAL and the counter need on occasions to exchange confidential data. Key material is supplied by the BAL to the counter and some confidential data is supplied by the counter to the BAL. Encryption of individual XML fields is used to protect the data (SSL cannot be used since the SSL session is terminated at the data centre network layer rather than the BAL).

Certain data in the XML messages is more sensitive than other data. This includes banking related information as well as user passwords.

User passwords are never sent across the interface, instead an SRP password verifier is sent when required, this is a part of the secure SRP protocol but is also encrypted.

Banking information that must be sent securely is encrypted by the counter PIN pad before transmission. This information is passed on by the BAL to the authorisation agents and is not decrypted in the BAL (it is not possible for the BAL to decrypt it). This information is not logged in the message journal table either. In general only a small part of banking data is encrypted (such as 'PIN blocks'). The rest of the data including PAN and Track 2 is passed through the BAL in clear. This is the main reason that that the BAL must treat data being passed through the online service routing component as sensitive and must not log any of the data to diagnostic logs.

Details of the encryption and security mechanisms are covered in the HLD documents.

2.3 External environment and constraints

2.3.1 Load Balancing

The BAL application servers will be load balanced in a round-robin (or similar) way at the network level before entering the Java environment. The servers will be designed to be stateless, but there will be some caching to improve performance and reduce the load on the BRDB. There will be no use of "sticky sessions". Wherever session state needs to be shared, this will be done by means of the Branch Database.

To make configuration and management easy, the architecture will not require clustering or any similar functionality. Instead it will accommodate the load balancing characteristics, making Services atomic and stateless, and having the servers only store and share minimal state (such as active logged on sessions) in the Branch Database and local caches derived from data in the Branch Database, where appropriate. Load Balancing between BAL and Branch Database will be based on the users' branch, therefore all transactions from the same branch will always hit the same database node.

Details of the database architecture can be found in HNG-X Branch Database Architecture (ARC/APP/ARC/0008).

2.3.2 Network characteristics

There are two primary characteristics of the network to take into consideration:

- There is no end-to-end SSL traffic: SSL traffic ends at the entry point to the Data Centre network. This means that the network cannot be considered entirely safe end-to-end, and we cannot rely on SSL solely to safeguard sensitive data such as key material or passwords.
- There are several layers of "indirection" through routers and firewalls: this means that we cannot easily rely on a route back to the counter from the server, therefore traffic between counter and server will have to rely on a traditional http request-response pattern of messaging.

These two characteristics have been, and will have to be taken into account in the rest of the architecture.

2.3.3 Other characteristics and constraints

Another major constraint in architecting the HNG-X system is that we will be using some backend systems that are outside our control. This means that we cannot rely on these systems to be up and running 24/7, neither can we rely on them always responding in a timely fashion. For these reasons the system will have to be architected with these characteristics in mind, when it comes to resilience aspects. Therefore we will have to achieve service level isolation, enabling services to run uninterrupted, even if a single service is temporarily unavailable due to backend systems not being available. Service Isolation is addressed in detail in section 2.4.1.

2.4 Service Interface Definition

The Counter Business Application used in the branches uses the Branch Access Layer as its server. Since the Counter is essentially a client of the BAL, the BAL needs no explicit knowledge of the Counter; it only needs to provide the Counter with a well defined interface. Strictly speaking this is not entirely true as for security reasons we do verify the configuration of the counter at login to ensure that the counter is a valid counter. This is done by referencing estate management information stored in the BRDB.

Furthermore, as the choice of transport protocol is HTTP, information has to be provided as to which service a request will be routed to.

A Service interface will therefore define three things:

- The endpoint at which the specific service is available (including required version of the service).
- The valid and expected format of the request.
- The valid and expected format of the response.

2.4.1 Service endpoints

A Service endpoint is a common term in the Web Services and SOA world for an address or location where a specific Service is available. The endpoint defines which Service a request should be routed to.

In the case of XML over HTTP an endpoint may take the form of http://somehost.com/services/helloWorldService.

The first portion defines the actual hostname of the host handling the requests. */services/* refers to the point at which the service routing component is mounted on the host. Finally *helloWorldService* defines the actual name of the Service to be used. In other words, the example above shows the endpoint address to be called to call a helloWorldService service.

This in its entirety makes up the Service endpoint. This is the address of which a client will need to be aware to call a specific Service to get a specific task executed on the server side.

Another benefit of using the URL to define the Service endpoint is that it enables service routing to occur based on a path called, rather than data in an XML message. This means that we can route service requests without incurring the performance hit of parsing XML, and deferring XML parsing until the point where it is absolutely necessary, and only do parsing once instead of several times.

Furthermore, it enables us to version Services, letting us run several versions of the same Service, but with slight variations, on different endpoints. For instance one could have one endpoint at */services/2009/11/helloWorldService*, and one at */services/2008/05/helloWorldService*, implying one version of the *helloWorldService* that was deployed in November of 2009, and another deployed in May of 2008.

2.4.2 Format of requests and responses

The protocol between the Server and Counter is to be XML over HTTP. The BAL-Counter interface has been designed with flexibility and ease-of-implementation in mind. We are not using a standards based protocol, but a simpler design based on an automated translation from Java objects to XML. This means that there is no interface specification that defines each individual message; the messages are defined by the Java objects.

The individual messages will be documented in a message catalogue which is still to be created, and their contents will be documented in HLD and LLD documentation.

The BAL-Counter interface can be considered a tightly coupled interface between two processes; it is not suitable for communicating with other processes or for integration into an ESB. However it is also required for the Audit subsystem so it can't be considered to be a private interface between Counter and BAL.

As this interface is shared with the counter, the counter side of the interface is documented in HNG-X Counter Business Application Architecture (ARC/APP/ARC/0009).

2.4.3 Conversational state

For all services there will be no conversational state between invocations, services will be stateless and atomic. This is in line with the Post Office requirement that the system be built according to SOA principles. It is also essential for meeting the system message volume requirements.

To add state to BAL servers would require some sort of state replication amongst the active servers. This is a complex thing to implement and is usually avoided in enterprise architectures.

There are a small number of cases that could be considered as exceptions to this rule, notably the login process which proceeds in two stages. The state that is maintained between these stages is stored in the BRDB.

2.4.4 XML Definition Best Practices

2.4.4.1 Handling of errors, exceptions and warnings

Propagating errors and exceptions occurring on the server side to the client side in the form of Java-specific errors, or http-specific errors should not be permitted. This is due to a couple of factors:

- Consistency: It is advisable that the Counter can expect to receive errors exceptions and warnings in a consistent manner, being able to recover gracefully rather than in an undefined manner. Also, a client application expecting an xml response should get an xml response, not an undefined string or http error.
- Application lifecycle: It is possible that the server side and Counter application will have different life spans. If say the counters life-span is shorter than the servers', and the counter application is redone or ported in another programming language, it does not make sense for it to try to understand arcane Java errors.
- Security: Propagating the text to errors deep down in the server software may be potentially unsafe, revealing implementation detail of the server software to eyes who should not know about it.

For these reasons, all response xml documents will follow the following format or something close to it (see Figure 2):

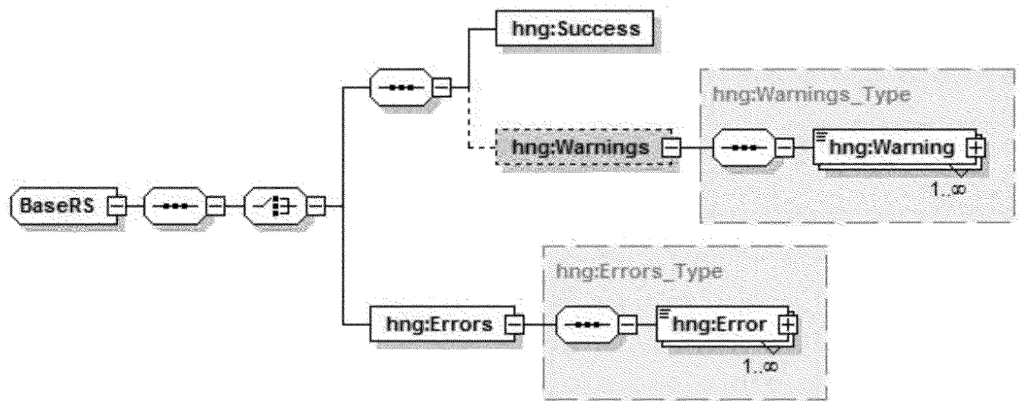

Figure 2 – XML schema defining a base response type

The above picture shows a graphical view of an XML schema defining a base response type (excluding potential future identity related data), it has the following characteristics:

- It has a choice, it will display either a *Success* element OR an *Errors* element, never both.
- The existence of a Success element indicates that the response returned successfully and no problems where encountered.
- When a *Success* element exists, it may be accompanied by an optional *Warnings* element, containing 1-*n* number of *Warning* elements. A Warning may contain information about the request that the client should be aware of, but which are not errors or critical to the processing of the request. An example of a warning might be if a Service doing some calculation has rounded the number of decimals provided by the client in the request.
- The *Errors* element may contain *1-n* number of *Error* elements. The existence of an Errors element indicates that the request failed and did not process properly.

When an error is returned the framework ensures that no database updates are made (rollback) and no message journaling is done. There should also be no state changes in any related internal or external system.

2.4.4.2 Implications on server-side implementation

The implication of this approach is that any Exception occurring on the server-side must be handled and either properly recovered from, or translated into an appropriate Error element form. Any Exception that is not handled in this way, and manages to sneak through must be considered a programming bug.

Care will be taken to include sufficient correlation ids in diagnostic logs to ensure that errors can be traced.

2.4.4.3 Parsing and building XML documents

As mentioned, the XML structure is derived from shared java objects. This common code shared between the counter and the BAL layer ensures that messages are interpreted correctly between the layers.

Considering that most requests will have some common structure, and most responses will have some common structure, it is likely that there will be ample opportunities for reuse of interfaces and abstract base classes, implementing common functionality. Prime candidates for common functionality are the requirements around digital signatures, identities and Error handling.

2.5 Software architecture

The software architecture will be composed of two distinct sub-architectures:

- Online Service Routing: for transformation and routing of requests going to "third party systems", such as banking, DVLA etc.
- Local Data Access Services: for services that primarily only use the database and other local or near-local services.

This does not however mean that there will not be some overlap between the two: some components, such as Branch Session Management and Service Invocation Framework, are likely to be shared between the two, although it may take the form of deploying the same software component and configurations in two places.

2.5.1 Service isolation and resilience

For resilience purposes, the system will need to be able to run services in relative isolation from each other. The degradation or even loss of one service should not adversely impact any other services. To achieve this we can configure individual queues within the BAL for different classes of services. By altering queue parameters we can alter service priority within an OSR server.

Timeouts are also used as an important part of server management. If a request is queued for too long it will timeout and be removed from the queue. Long running tasks such as reporting will have larger timeouts and larger queue lengths to reduce the chances of a timeout on these tasks.

2.5.2 Service Invocation Framework

Figure 3 below is a high-level view of the Service Invocation Framework used for both Local Data Access Services and Online Service Routing. The Service Invocation Framework has been designed to run within the Interstage Servlet Container, and also within the Online Service Router. This leaves open the future option of hosting the Service Invocation Framework within any number of application servers, including but not limited to Interstage.

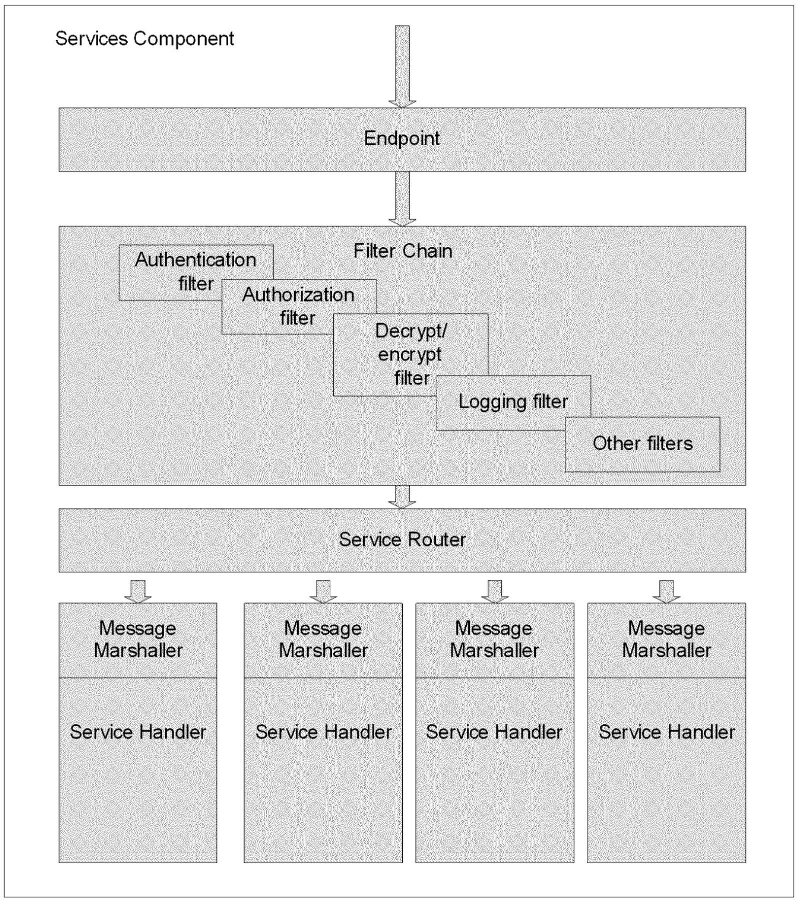

Figure 3 – Service Invocation Framework architecture

- **Services Component:** The overarching deployment package for the Services architecture
- **Endpoint:** the endpoint is a protocol-specific endpoint, such as an HTTP Servlet, or Message Driven Bean, which receives the protocol specific message and translates it into a protocol independent InvocationContext-type object, which the layers below can manipulate without knowledge of protocol specific details.
- **Filters and Filter Chain:** After the Endpoint, the InvocationContext can be run through a chain of Filters. The filters have the responsibility to perform any cross-cutting concerns and pre-processing tasks required before a Service is invoked. This can be things such as authentication, authorisation, decryption and encryption of specific XML fields etc. The Filters can be “chained” in different configurations depending on needs, so that they are executed in a logical sequence of events. The filters follow a traditional “Pipes & Filters” pattern of implementation.
- **Service Router:** The responsibility of the Service Router is to route an invocation to the correct Message Marshaller/Service Handler pair. The Service Router follows a “Message Router” pattern where routing is based on Service name.
- **Message Marshaller:** The responsibility of the Message Marshaller is to parse a request into a format that the Service Handler can understand, and then after the Service Handler has

executed, marshalling any potential response from the handler back into an agreed upon message format (such as XML). In the proposed architecture the Message Marshaller is a single component across all Services, with the ability of parsing XML into DTOs and back.

- **Service Handler:** The Service Handlers follow a traditional "Command pattern"; they take input from the Message Marshaller and execute. The Service Handler encapsulates the business logic of performing a specific Service. It is likely that a Service Handler in itself will have a more fine-grained model, involving a Domain Model, Data Access Objects and other lower-level concepts. However, this is not part of the Services infrastructure architecture, and therefore not detailed in this description.

2.5.2.1 Designing and implementing a Service

Designing and implementing a Service should follow the following steps:

1. Define the Service interface with Java Data Transfer Objects
2. Configure the Message Marshaller
3. Implement a Service Handler that executes the business logic of the Service based on the inputs of the Message Marshaller
4. Publish the interface specification, and configure the Message Marshaller and Service Handler pair into the Service Router so that the Service becomes accessible.

2.5.2.2 Notable characteristics

The Services architecture is protocol-independent and symmetrical. It is feasible to have the same basic infrastructure framework (Services Component), but in different configurations, sitting in a hierarchy. For instance we may have a top level deployment that simply forwards an invocation through something like a JMS Queue, where a Message Driven Bean in turn acts as an Endpoint and forwards the invocation to another set of Filters and a Service Router. This may or may not be needed, but it is possible and gives us the flexibility of choice. It is also deployable both within the Interstage Application Server, as well as the Online Service Router component.

2.5.2.3 Service Error/Exception conditions

There are a number of possible error conditions when invoking/using a Service. However, the isolation of these conditions is essential; therefore it is important that we do not have any "Exception leaks" where unexpected exceptions leak up through the architecture. For this reason we will define the valid Exceptions that may occur during execution of a Service:

- Parsing exception: This would occur in the case of a request message being in an invalid format. This will occur in a Message Marshaller.
- Validation exception: The request message is in a valid format, but has invalid values in it. This would occur in the Message Marshaller.
- Connectivity exception: This happens when a Service Handler is unable to use or connect external resources such as a database or messaging system that it needs to use to perform the Service.
- Service exceptions: These are exceptions that occur when performing a Service within a Service Handler. These types of exceptions should include business logic based exceptions.

2.5.3 Services Architecture for Online Service Routing Services

The BAL OSR has been created in order to manage high transaction throughput to diverse 3rd party systems whilst providing a high degree of service isolation. A non blocking I/O solution has been adopted that can operate using queues that are processed by a small number of "worker threads". The solution is immune to different response times from different services and performance of one service cannot impact on another. This provides a high throughput solution that is both scalable and extensible.

The Online Service Routing architecture will not run within a J2EE container. This is due to two factors:

- Use of a custom HTTP multiplexer: This utilises custom threading and IO code to efficiently manage resources during peak server loads.
- Custom NIO connectivity framework: This is the most efficient way for Java to handle socket based communications.

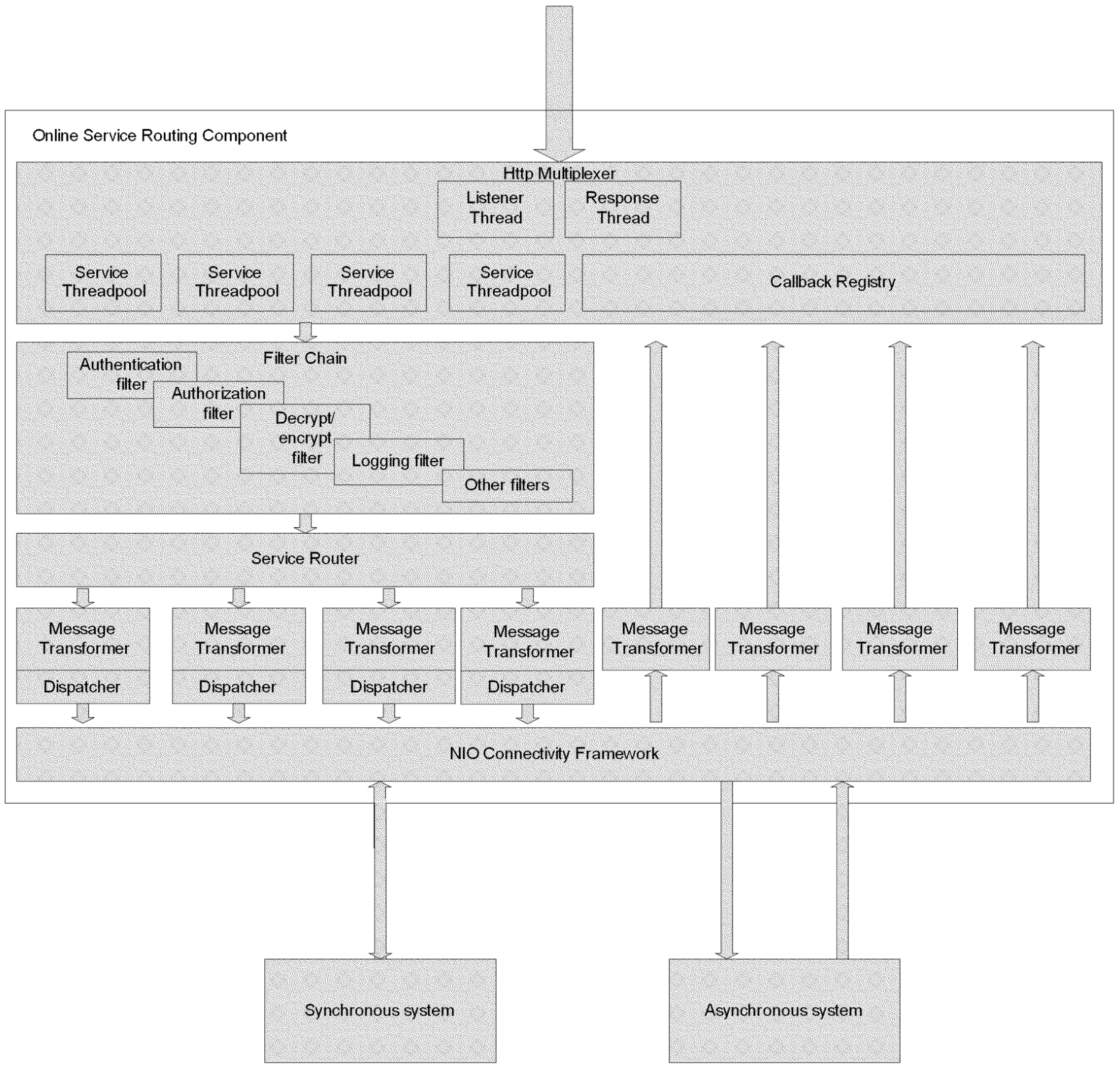

Figure 4 – Online Service Routing architecture

- **NIO Http multiplexer:** This is a custom lightweight HTTP server that will be implemented for the specific use of the Online Service Routing. It provides a non-blocking, high concurrency and high throughput http server that allows for custom resource allocation and service isolation. Its architecture will be described in more detailed in the next subsection.
- **Filter Chain & Filters:** this will be a common component with the Service Invocation Framework, and is described in the section for the Service Invocation Framework.
- **Service Router:** Has the responsibility of routing requests into the correct Message Transformer & Dispatcher combination. This is essentially the Service Invocation Framework detailed earlier, configured for use within the OSR.
- **(Inbound) Message Transformer:** Transforms an incoming message into the format expected by a third party system, such as DVLA or authorisation agents.
- **Dispatcher:** Has the responsibility of defining the target address and protocol which the NIO Connectivity Framework will call. The Dispatcher will also define the outbound Message Transformer that the NIO Connectivity Framework will call with the response, once a full response has been received.

  After the Dispatcher has finished, the dispatcher and request context will be put on a queue to the NIO Connectivity Framework. This in effect terminates the execution thread for a specific request.
- **NIO Connectivity Framework:** Will be described in more detail in a later sub section. The NIO Connectivity framework has the responsibility of handling the logic and semantics of the actual connectivity to backend systems. The Framework also optimises resource use by making efficient use of threads. One thread can service multiple input or output requests at a time and is not blocked while waiting for data. This means that the framework will use drastically less system resources than if each request was assigned a thread that did an outbound connection.
- **(Outbound) Message Transformer:** In essence the same interface as the inbound transformer, with the difference that the particular implementation is adapted to create counter responses from backend system responses. Once the outbound Message Transformer has finished, a callback thread will call back with the response into the callback registry of the http listener, which in turn will wake up a thread that writes the response back to the client.

2.5.3.1 NIO-based HTTP Multiplexer

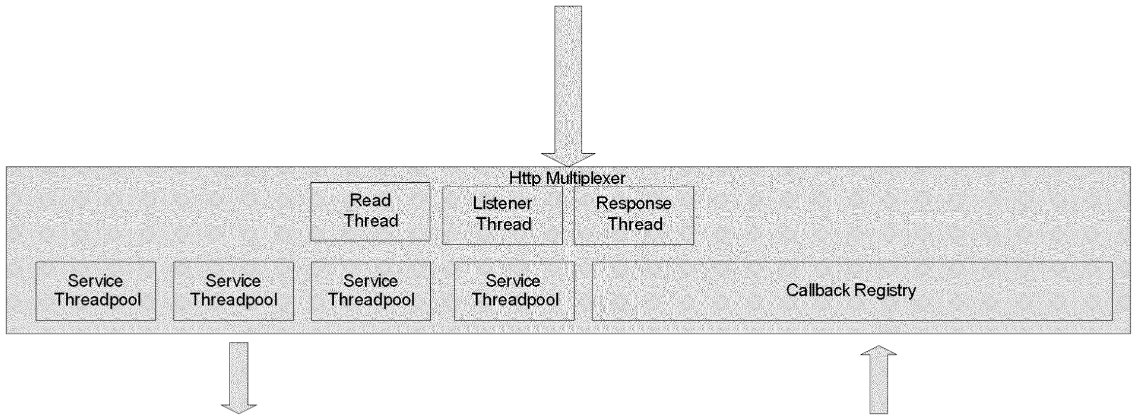

Figure 5 – NIO-based HTTP Multiplexer

The NIO HTTP Multiplexer makes use of Java Non-blocking I/O APIs (NIO, Java NIO or java.nio). This allows the multiplexer to be notified of I/O events, such as a new connection, while not blocking other clients connecting while doing so. This alone increases the theoretical concurrency compared to Apache considerably.

The multiplexer also enables isolated, service-specific thread pools, which means that it enables true service isolation.

These are the other characteristics of the multiplexer architecture:

- **Listener Thread:** This thread listens for and accepts new connections, before passing connections onto a read queue.
- **Read Thread:** picks requests off the read queue and reads them into a network-independent format, before putting the read request onto a specific service thread pool. The read thread also allocates the request a correlation id, and puts the underlying connection with the correlation id into the callback registry.
- **Service Threadpool:** Separate thread pools may be allocated, based on service endpoint URL patterns; for instance one thread pool may be assigned to "/DVLA/*" and one for "/banking/*". Each thread pool may then be allocated an individual number of threads. This helps the multiplexer achieve service isolation. Once the read thread is done reading a request, it will interpret the request, put it onto a queue corresponding to the thread pool of the requests URL, from which the specific thread pool then picks the request up when it is able to process the request. Once a Service Thread has executed, a request is no longer associated with a thread; instead, the response will be written to the connection whenever a callback comes into the callback registry.
- **Callback Registry:** The callback registry holds connections and their respective correlation ids until a callback is received from the outside or until a timeout occurs.
- **Response Thread:** Once the callback registry receives a response, the response is matched to an entry in the registry through the correlation id, and the response is written to the connection before the connection is closed.

2.5.3.2 NIO Connectivity Framework

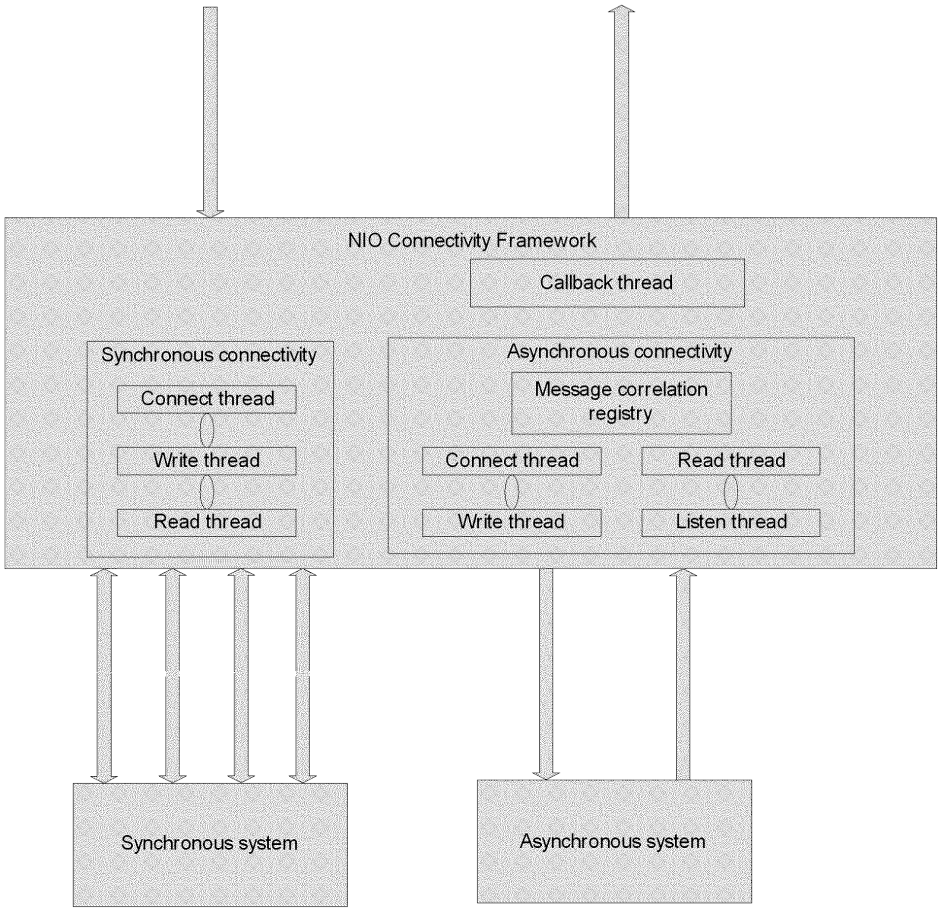

Figure 6 – NIO Connectivity Framework

The NIO Connectivity Framework works on the principle that in order to avoid blocking on network waits, such as connecting and a connection being readable or writeable, the framework will break these steps into separate individual stages.

2.5.3.2.1 Synchronous connectivity

For synchronous connectivity, a request will have to provide a callback object used by the callback thread, and go through the following steps:

1. Being on the connect queue
2. Initialise connection by the connect-thread
3. Be put on the write queue
4. Connection being written to by the write thread
5. Being put on the read queue
6. Connection being read by the read thread

7. Response with callback object being put onto the callback queue
8. Callback object being called by the callback thread.

2.5.3.1.2 Asynchronous connectivity

Asynchronous connectivity will go through steps similar to those of the synchronous connectivity; the main difference here is that we will have a message correlation registry to correlate requests with responses, and a need to multiplex on the open connection to differentiate responses from requests.

This is a very flexible design that can accommodate many types of communication protocols. Protocols such as the authorisation agent interface maintain multiple open connections to back end servers (active and passive for example) and can switch connections based on authorisation agent notifications or connection errors. Routing to individual back end nodes can also be implemented at this point based on load balancing rules or message content.

2.5.4 Components to Architecture mapping

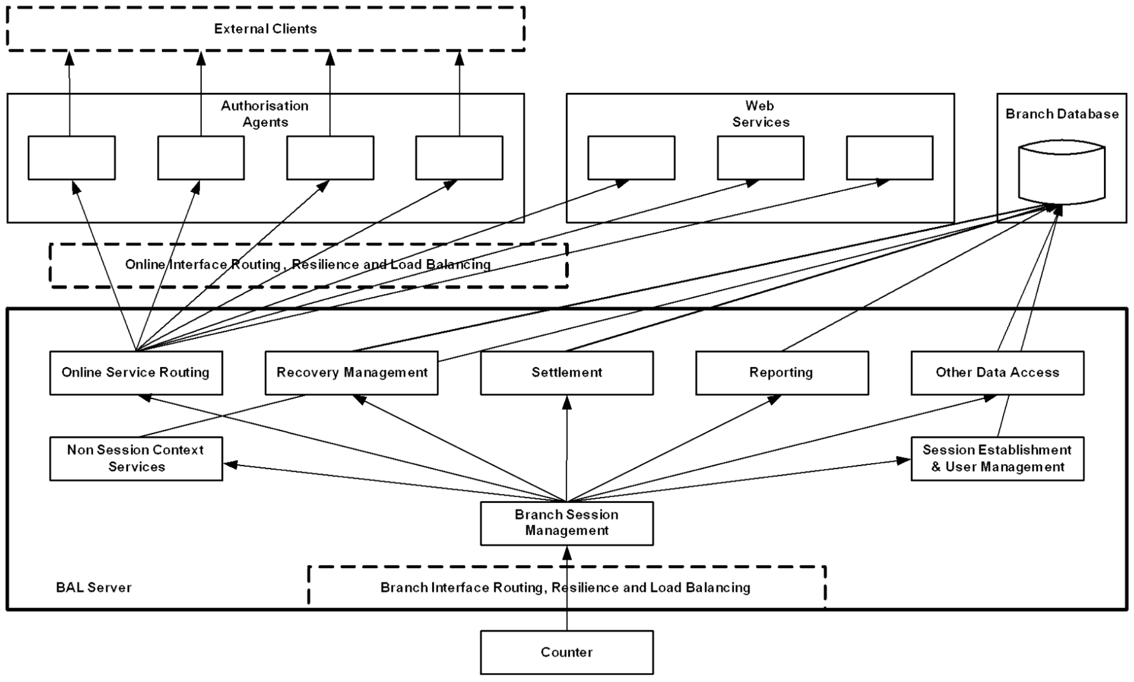
External Clients
Authorisation Agents
Web Services
Branch Database
Online Interface Routing, Resilience and Load Balancing
Online Service Routing
Recovery Management
Settlement
Reporting
Other Data Access
Non Session Context Services
Session Establishment & User Management
Branch Session Management
BAL Server
Branch Interface Routing, Resilience and Load Balancing
Counter

Figure 7 – Components to architecture mapping

The Services Component maps to what is within the "BAL Server" in the above diagram. The components within the Application Server maps as follows:

- **Branch Session Management:** The branch session management maps to the Filters & Filter Chain concept described in the Service Invocation Framework. This is a shared software component between the two architectures (see 2.5.2).
- **Online Service Routing:** Online Service Routing maps to the architecture with the same name detailed in section 2.5.3.

- **Recovery Management, Settlement, Reporting, Other Data Access:** These components map straight to Message Marshaller/Service Handler pairs in the Service Invocation Framework. However, it should be noted that it is possible and very likely that each component will be made up of several individual Service Handlers for the different use cases/services for each component.
    1. **Settlement:** Settlement will occur only once for every customer session, and will use a ServiceHandler to write a settlement record into the Branch Database amongst other actions.
    2. **Recovery Management:** For certain transactions, such as Banking, a Recovery record will be written in correlation with a Service request for the specific service. If a failure has occurred, Recovery will be invoked on the next login.
    3. **Reporting:** Reporting has its peak just before the end of core service hours. This will use a Service Handler to return all Report data to the Counter.
    4. **Other Data Access:** depending on the context, this will in most instances use a simple MessageMarshaller-ServiceHandler pair as per standard service implementation.
- **Non Session Context Services:** Same as above. The difference in this case is likely to be that the Branch Session Management filters will be configured not to run on these services, but instead just pass the invocation through.
    1. The counter application uses the non session context services to obtain information from the branch database when there is no user logged on, for example to obtain reference data updates. The set of services provided in this way will be strictly limited. The full set of services will be decided within the design phase.
- **Session Establishment & User Management:** These map to Service Handlers as well, but in the case of Session Establishment, we have to configure the BSM filters to pass through the requests rather than authenticate and authorise them.

2.5.4.1 Help

Counter help pages are not retrieved from the BAL, they are updated by the reference data distribution. At one stage it was planned that the BAL would provide the help text but this has now changed.

2.5.5 Authentication/authorisation architecture

2.5.5.1 SRP

We use the internet standard SRP protocol revision 6a for login authentication. The SRP protocol has a number of desirable properties: it allows a user to authenticate to a server, it is resistant to dictionary attacks mounted by an eavesdropper, and it does not require a trusted third party. It effectively conveys a zero-knowledge password proof from the user to the server. Only one password can be guessed at per attempt in revision 6 of the protocol. One of the interesting properties of the protocol is that even if one or two of the cryptographic primitives it uses are attacked, it is still secure.

The SRP protocol creates a large private key shared between the two parties in a manner similar to Diffie-Hellman, then verifies to both parties that the two keys are identical and that both sides have the user's password. It is also independent of third parties, unlike Kerberos. The SRP protocol, version 3 is described in RFC 2945. SRP version 6 is also used for strong password authentication in SSL/TLS and other standards such as EAP and SAML, and is being standardized in IEEE P1363 and ISO/IEC 11770-4.

For more information on SRP see http://srp.stanford.edu/ .

2.5.5.2 Establishing a logon

This section provides an overview of the process, for detailed information please see HNG-X End-To-End Key Usage Overview HLD (DES/APP/HLD/0094), or HNG-X Online Service Routing High Level Design (DES/APP/HLD/0050). The following sequence describes the counter login process.

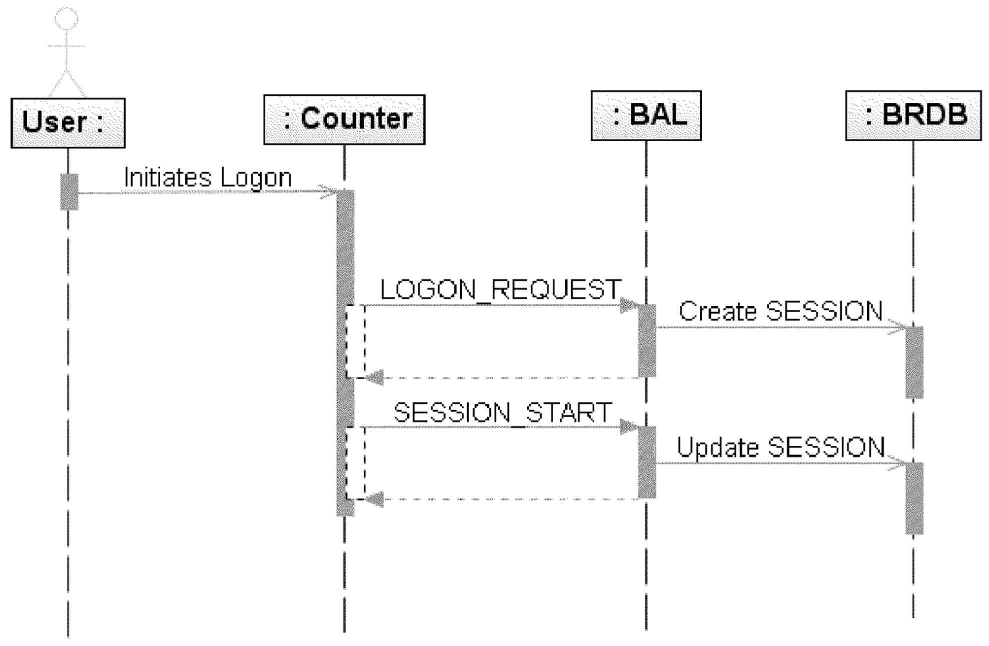

Figure 8 : Counter Login Process – Sequence Diagram

This corresponds to a two-round-trip SRP implementation rather than the basic three-round-trip design.

SRP uses a 'verifier' rather than storing a hash of a user's password. The verifier is stored in the BRDB and is never sent across the network. A random 'salt' is added to the password in the verifier creation process to complicate cryptographic attacks such as dictionary attacks.

The SRP verifier and the salt are stored in BRDB_BRANCH_USERS table. LOGIN_REQUEST sends the username and the calculated SRP values to the BAL. The counter session public key is also stored here to allow verifying later session messages.

The BAL responds with the user's salt and other required SRP values.

SESSION_START is used to authenticate the counter to the BAL and the BAL to the counter.

A random session key (called TokenId) is generated to identify the session. This is used only as a key to lookup the session information in the BRDB, it is not security related. The authenticity of the session is verified by checking the message signature that is sent from the counter with each message and by checking that the message header values match the values stored in the BRDB (such as source IP address, counter user and training flag).

Further to this, the logon process will be protected at a network layer through SSL between the counter and the data centre, as will all other traffic.

The BAL will check the counter IP address against estate management information in the BRDB. If the IP address does not match the registration information the login will be disallowed.

Each transaction with the BAL rechecks various header values to ensure they match the login session values. This includes the BranchID, CounterID, Training Flag, and UserID.

The change password process sends the encrypted password to the BAL; this is required as we need to check this password against previous password values. Details of this process can be found in the BAL HLD document.

2.5.5.3 Authorisation model and maintaining a session

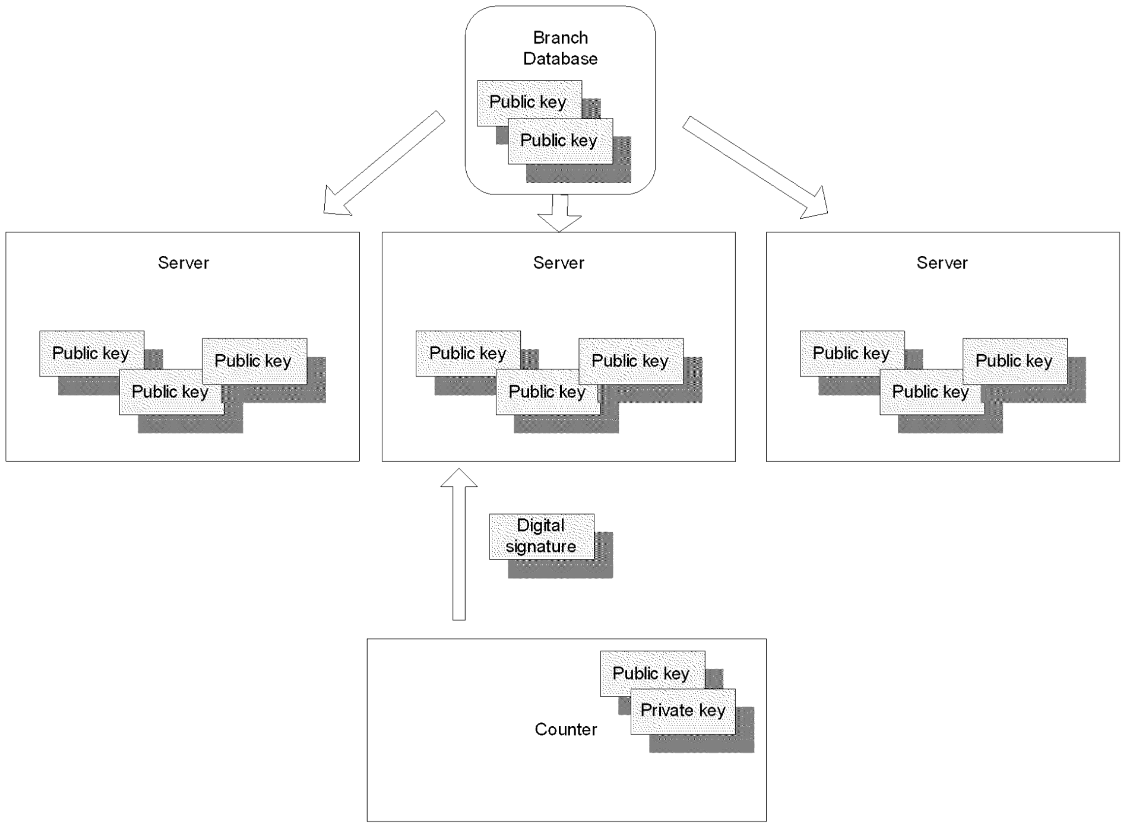

Figure 10 – Authorisation architecture

The diagram above reflects the major flows and components of the authorisation architecture:

Authorisation

a. A User sends a request to the server, with a username and signature derived from signing the message body or subset of the message body.

b. The server looks up the corresponding session public key from the BRDB.

c. The server verifies the signature and authorises the request for further processing.

Logoff

1. The User sends a logoff request with a signature and username
2. The server verifies the signature as above.

3. The server marks the session as deleted in the branch database

Forced Session Termination

1. The server identifies a requirement to force terminate a session. E.g. an active user attempts log on at a different counter position.
2. The server marks the existing session as deleted in the branch database.

Serverside Authorisation in more detail

For the most part, the counter will police what services a user can and cannot use. However, in some cases, such as user management, we will need to police this on the server side as well. For this reason we will have to take the following approach to authorisation:

- Have a list of services that require further authorisation checks, and match each request against that list.
- If a request is on that list, retrieve the role of the user making the request and see if that role has privileges to access that particular service.

It is likely that we will store this data in the database, but cache the most common things requested to minimise data access.

2.5.5.4 User – Role – Service privilege relationships

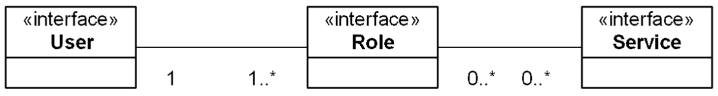

Figure 11 – Authorisation – conceptual model

The above diagram describes the conceptual model of authorisation: A User has one or several Roles, where those Roles in turn have privileges to use defined Services.

Since the counter will police most of the access, this is not necessarily how the relationships will be represented in the Data Centre, other than for those Services that require further server-side authorisation.

As a whole, the conceptual model for authorisation is more or less the standard approach taken by most authorisation implementations in the market.

Details of the authorisation rules are provided in HNG-X Online Service Routing High Level Design (DES/APP/HLD/0050)

2.5.5.5 Key Exchange

The following cryptographic keys are obtained by the BAL: Note that a full explanation to the keys used can be found in DES/APP/HLD/0094.

- PAN HASH SEED – this key is used in the counter as the seed for the hash algorithm used to create a hash of a PAN. This enables transmitting hashes rather than sensitive customer data. There is a single global PAN HASH SEED. This is obtained from the Key Server
- BAL JOURNAL SIGNING KEY PAIR – the BAL is required to sign the login journal entry stored in the Message Journal table. The private key of this key pair is used for this purpose. The certificate of the signing key is included with the signed message. This is obtained from the Key Server
- COUNTER SESSION KEY PAIR – a new key pair is generated for each session by the counter.

PASSWORD ENCRYPTION SYMMETRIC KEY - used for encrypting the password verifier and salt for storage in the BRDB. This key is obtained from the Key Server. A temporary symmetric encryption key is used for transmission of the password to the BAL during the change password process.

2.5.6 Database Access architecture

2.5.6.1 Parameterised queries

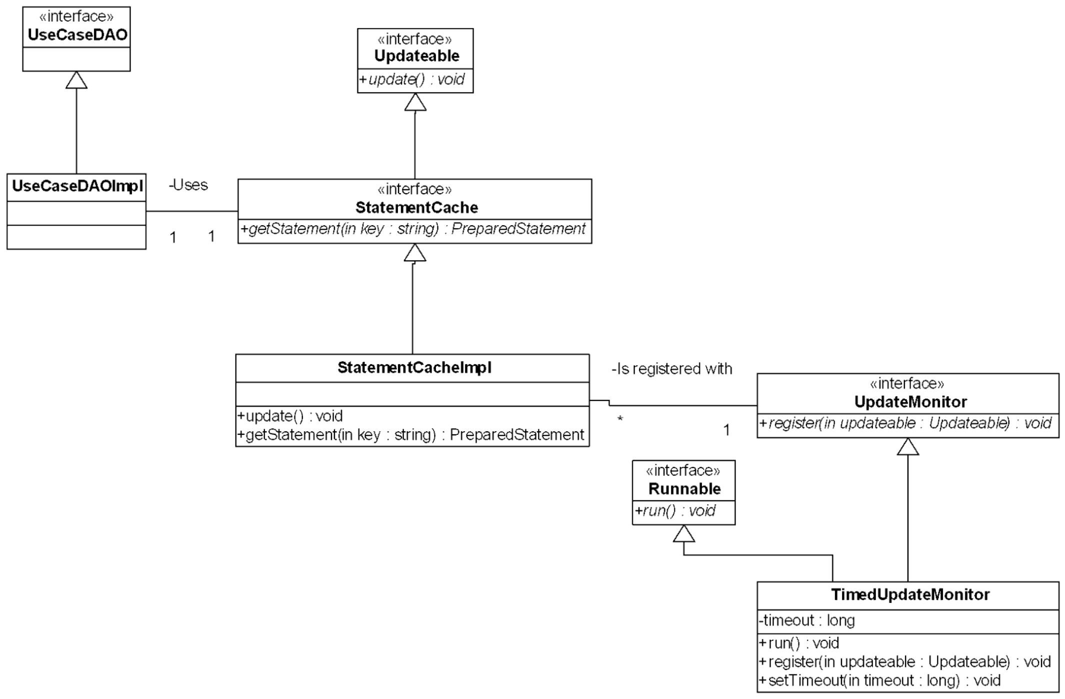

Figure 12 – Parameterised queries structure

There is a recommendation for *PreparedStatements* to be derived from a database-table controlled by the DBAs. This means that any database access will have the following characteristics (see conceptual figure above):

- Data Access Objects (DAOs), such as the conceptual *UseCaseDAOImpl* above will use a *StatementCache* object to retrieve *PreparedStatements*. The *PreparedStatements* will be retrieved by means of a key, such as “CUTOFF_REPORT”.
- A *StatementCache* implements the higher level *Updateable* interface, which marks an object as being updateable by some outside event. The *update* method implements the actual update logic, such as reading in new versions of *PreparedStatements*.
- The actual implementation of the standard *StatementCache*, *StatementCacheImpl* implements all the *update* and *getStatement* logic. During runtime, the *StatementCache* object will be registered with an *UpdateMonitor*.
- An *UpdateMonitor* is an object that monitors for updates, and will update any *Updateable*-objects that may be *registere*d with it.

- For the *StatementCache*, we will by default use a *JMX* MBean that will enable the external monitoring tools to start a cache refresh if required.
- For the StatementCache, positional parameters must be used to prevent SQL injection attacks.

2.5.6.1.1 Noteworthy characteristics of parameterised queries

There are some things that should be noted with regard to parameterised queries:

- Queries (and results) have named parameters so the order of parameters is not important. It is important though that the query template matches the parameter specification.

2.5.6.2 Matching a DataSource with a counter/branch (DB Load balancing)

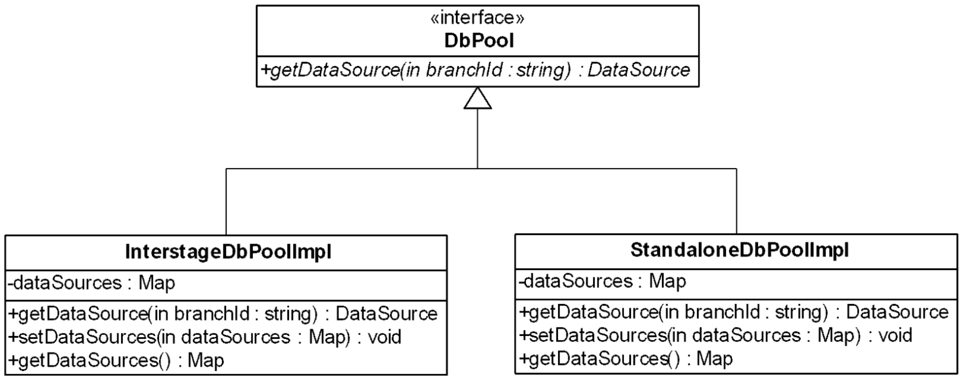
«interface»
DbPool
+getDataSource(in branchId : string) : DataSource
InterstageDbPoolImpl
-dataSources : Map
+getDataSource(in branchId : string) : DataSource
+setDataSources(in dataSources : Map) : void
+getDataSources() : Map
StandaloneDbPoolImpl
-dataSources : Map
+getDataSource(in branchId : string) : DataSource
+setDataSources(in dataSources : Map) : void
+getDataSources() : Map

Figure 13 – DataSource retrieval structure

Load balancing over the different nodes of the Oracle cluster will be done by table lookup from the BRDB within the BAL-application based on the branch-id (and hence FADHash) of the requesting counter. To hide the implementation logic around this, we will implement our own DbPool interface, which will live as a singleton within the JVM it is running in.

The DbPool implementation will encapsulate the actual looking up of the correct DataSource. Each BAL Server instance will have 6 actual database pools, one for each of the 4 Oracle databases and an additional datasource that is used for bootstrap purposes plus a training datasource. The bootstrap datasource is used to retrieve the FADHash map that is used to map BranchIDs on to database instances; this datasource is connected to the Oracle RAC endpoint, whereas all of the other datasources are connected directly to an Oracle database instance. The bootstrap datasource is also used after a connection failure as the FADHash map may need updating to map out a failed database. Details of this mechanism can be found in the BRDB architecture documents.

2.5.6.3 Failover

When the BAL detects a failed database connection it will re-read the FADhash map using the bootstrap datasource. This map will have been updated by the BRDB to map out the failed database. This failover process will be complete within 10 seconds of the failure being detected.

An overnight process re-reads the FADhash map, so a failed database will be mapped out until this overnight process runs, even if it becomes available during the day.

If the bootstrap datasource becomes unavailable (an entire cluster failure) the BAL cannot continue processing messages from the counter. In this case the BAL will reject incoming messages while at the same time continually retrying to connect to the database. As soon as the database becomes available again the BAL will resume processing messages.

2.1.7 Deployment architecture

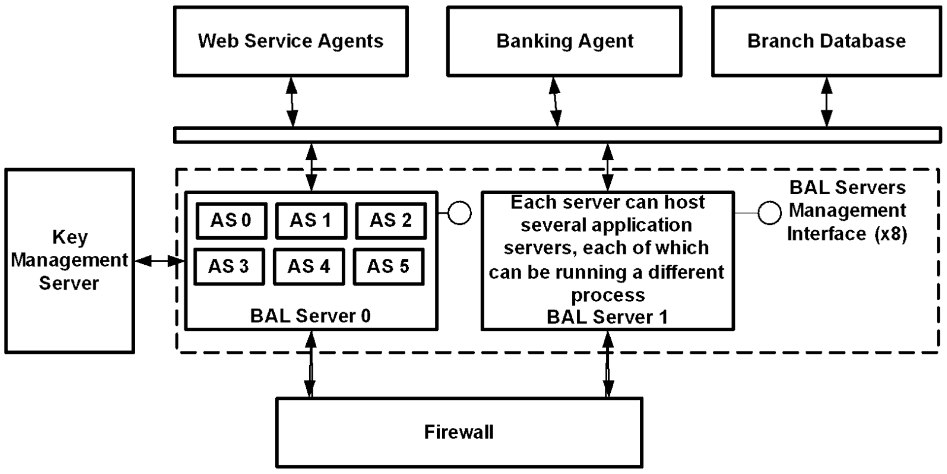
Web Service Agents
Banking Agent
Branch Database
Key Management Server
AS 0
AS 1
AS 2
AS 3
AS 4
AS 5
BAL Server 0
Each server can host several application servers, each of which can be running a different process
BAL Server 1
BAL Servers Management Interface (x8)
Firewall

Figure 14 – Deployment architecture

The Online Service Routing Architecture (section 2.5.3) (shown as AS in the diagram) is deployed as a standalone Java process/daemon and makes use of the Service Invocation Framework. Earlier designs included a J2EE application server alongside the OSR but this has now been removed. This is due to the amount of custom I/O and threading code in the architecture, and its lack of fit within a standard J2EE environment. Deploying it as a standalone Java process/daemon gives us the best flexibility in developing the architecture.

The diagram shows that multiple OSR instances can be deployed within one physical/virtual server. This is to provide resilience and to make the best use of available resources. Each BAL server has a large amount of RAM (up to 8 GB), the best way to utilise this RAM is by deploying multiple Java VMs as a single Java VM will only efficiently use a few GB of RAM. The exact mix of instances will be determined by performance testing once the code is available.

Online Service Routing deployment and maintenance is supported by the use of JMX where applicable, and requires custom management/deployment tools/methods where JMX is not sufficient.

3 Functional Characteristics

The BAL provides many services to the HNG-X counter. These services are not described in this document; in general they are derived from the use case documents and form a part of the business processes that the system is designed to accommodate. However some services are of a generic nature and therefore it is useful to describe them here.

The complete list of services is described in BAL Interface Specification (DES/APP/IFS/0012).

3.1 Services are stateless

In general services will be stateless and self-contained within one request/response exchange. Any conversational state for individual services will be designed and maintained on a case-by-case basis by means of the Branch Database. This will also avoid the need for any server affinity (sticky sessions).

This assumption is consistent with the Post Offices requirement that the architecture is to be according to SOA principles.

3.2 Generic Services

Some services are designed for generic data centre access. These services are described below.

3.2.1 Generic Read Service

This service provides the capability for the counter to read data from the Branch Database. The name 'Generic' is possibly a little misleading as this service does not allow the reading of arbitrary data.

The counter provides a key which describes the data that it wishes to retrieve. This key is used to look up the actual query in the database. There is additional metadata that describes the permissible input parameters that can be used to filter the data.

Generic Read is controlled by Role Based Access Control at the level of the individual query, i.e. the system can be configured to allow or deny access to any individual query based on the user's role.

Reporting makes use of the Generic Read service.

3.2.2 Generic Write Service

This service provides the capability for the counter to write data to the Branch Database. It works in a similar way to the Generic Read service described above.

3.2.3 Generic Web Service Interface

This interface allows the counter to send a request to a Web Service endpoint. The BAL does not need to know any details about the web service being requested apart from the address of the endpoint.

The Generic Web Service interface allows the deployment of new functionality to the counters by reference data updates only. The BAL does not need to be rebuilt or reconfigured (apart from adding the endpoint address in the case of a new service. An additional message to an existing service requires no BAL configuration).

The endpoint address, and hence the allowed web services, is stored in configuration data in the BAL. This is a security requirement: it ensures that a counter cannot connect to an arbitrary web service but can only connect to a web service that has been configured for access.

3.1.4 Training Mode

The HNG-X system supports training mode capability, but initially this is only available from dedicated Counter Training Offices (CTO). These branches are managed as normal branches within the Live estate, but transactions from these branches are isolated from Live data. The BAL provides two distinct functions in support of Training. CTO branches are identified as such within reference data. The BAL logon interface ensures that any CTO branch user logging on can only establish a session with the Training mode flag set. Secondly, the BAL ensures that Live and Training mode transactions are handled separately.

The BAL interprets a training mode flag on the header of incoming requests. If the training mode is valid (the user logged in using the same mode) then the BAL redirects all database access to the training database and redirects most online services to the training web service (PAF for example is not redirected). The redirection of web services is configurable by web service. Some services, such as PAF are not redirected during training; the live service continues to operate.

This allows a standard BAL to act as a server for training counters, meaning that the training software is always up to date with the current live software and no special mechanisms are needed to maintain and update training software.

There is some risk here that training data could get mixed with live data. This risk is very small though as there are a few points within the BAL that we look at the training flag and redirect information to different endpoints, there is no need to make changes to individual services, the changes are made at the framework level.

More information on the CTO design can be found in CTO Design (ARC/SOL/ARC/0005).

4 Platforms

4.1 Operating System: Linux

The target operating system platform for the BAL is Linux (RHEL4). The chosen operating system should have minimal impact on the architecture and code, as this is platform independent. However the operating system will affect some of the characteristics of the application.

For instance, the custom NIO HTTP multiplexers' capabilities will be more predictable on a Linux platform as opposed to a Windows platform, as Linux has a more predictable model for calculating the number of available sockets on the OS. Furthermore, in our custom threading code, thread priorities will be impacted slightly by the underlying OS, however if we limit the use of priorities to Java's three default priorities, high, low and medium, we should be able avoid any impact.

Installation and maintenance scripts will be platform specific and will target Linux only.

4.1.1 Tuning & Configuration requirements

The underlying Linux platform will have to be tuned and configured for two factors that we are aware of at this time:

- Network stack for the maximum number of sockets, so that the operating system can support a high number of concurrent network connections.
- The TCP wait timeout values: when a socket is disconnected, it will be in a state in which it is not released immediately, instead it will have a timeout, normally around two minutes before it is usable again. This timeout value will preferably be lowered somewhat.

Note: Some of these options may to some degree also be tuned by means of setting the correct parameter options for sockets in the application code.

4.2 Java Environment

The Java environment used will be Java 6. Initial development was done on Interstage Java 4, as was the counter development. Due to code sharing between the counter and the BAL, most code will be Java 4 compatible.

The topology or architecture of the networks does not affect or impact the HNG-X Branch Access Layer architecture in a significant way. The architecture functions with a round-robin load-balancer and non-sticky sessions, with each BAL node running in a stand-alone mode. The BAL is designed to be stateless to enable scalability and resilience.

However, in order to do this, a few things have to be noted:

- Load balancing is assumed to happen in front of the actual BAL servers. Other portions of the infrastructure will handle the load balancing of http requests and the logic concerning that.
- SSL is terminates before the Branch Access Layer is reached. This is particularly important to the Online Service Routing Architecture, because if SSL was not terminated, SSL would have to be implemented in the custom http multiplexer, which would increase the amount of work required on this component.
- The Network is expected to present source IP addresses to the Branch Access Layer.
- HTTP Keep-Alive will be set to false due to the concurrency volume.

- The Logon process will take place over two requests. Without sticky sessions this will require that the encrypted logon token will be persisted to a database for the second request to be able to reach it. This is the only stateful message pattern.
- A feedback mechanism for service availability will have to be implemented. This is part of the manageability aspect, detailed later.
- The BAL will need to be set up with appropriate firewalls between it and the branch estate and also between the BAL and the rest of the data centre components that it uses.

4.3 Fujitsu BladeFrame

The BAL servers will be hosted on a Fujitsu BladeFrame. Each server blade will be configured as a vBlade (virtual blade) but will occupy an entire pBlade (physical blade). There would be no benefit in deploying multiple vBlades on each pBlade, the operating system overhead would just reduce performance while at the same time increasing the system management overhead. Using a vBlade allows us to make use of the failover services provided by the BladeFrame while at the same time maximising the performance available to the BAL layer.

Each BAL server will run the Red Hat Linux operating system. On each server there will be deployed multiple OSRs as described in section 2.5.7.

Detailed BladeFrame design will be completed at the design stage of the project and is highly dependent on available hardware and BladeFrame capabilities.

Instances of BAL servers will be distributed over available BladeFrames to maximise resilience.

5 Networks

Networks do not directly impact the BAL any further than what has been addressed elsewhere in this document. For network related areas, please refer to the rest of the document.

5.1 SSL

Messages from the Counter are sent using SSL. The SSL connection is terminated at the Cisco ACE blade though, so the BAL does not see the SSL connection. This has a number of advantages for the BAL as the overhead of the SSL is offloaded to the network layer.

5.2 Load Balancing

As there will be multiple instances of the OSR on multiple servers there is a need for load balancing network equipment. The load balancer will work with a least-active-connections algorithm and will treat all instances as equivalent. The actual number of servers and instances will be determined during performance testing and is therefore not known at present. In terms of network setup we have defined 10 as the maximum number of servers and 8 as the maximum number of instances per server. It is likely that there will be only 2 - 4 instances per server, but the network is designed to cater for up to a maximum of 8.

Section 2 describes the deployment configuration in more detail.

5.3 Network Health Check

Each OSR instance will provide a simple health check interface for the load balancer. This interface will return an empty message with a HTTP 200 response if the instance is available for new connections. All other responses or no response at all mean that the instance is not available.

Further information on network configuration can be found in BAL Interface Specification (DES/APP/IFS/0012).

6 Manageability

6.1 Diagnostics logging, event alerting, performance monitoring

Logging, event alerting and performance monitoring will be performed using two major methods:

- **JMX**: for real-time monitoring of the application, performance and important events.
- **Syslog**: for general logging, diagnostics logging and logging of software events.

JMX MBeans will allow us to monitor the applications in real-time with the help of monitoring tools such as Centric Manager, Tivoli or other JMX enabled management and monitoring tools. JMX is a standardised Java API that is widely used for these purposes. The exact uses of MBeans should be decided on a case by case basis in the lower level designs.

For logging, we will use Log4J, which is an open source, but de facto standard logging framework for Java. It provides several useful functions, such as multiple outputs for logging (file system, network, database or e-mail), and can easily be configured to send out a notifications and alerts through a number of types of channels in the case of exceptional events. The log file will be sent to the UNIX syslog to be picked up by the data-centre monitoring tool (NetCool).

Note that Audit logging is a completely different process to what is discussed here. Audit logging is a business process and is not described in any detail in this document. Further information can be found in HNG-X Online Service Routing High Level Design (DES/APP/HLD/0050).

6.1.1 Service management & configuration

Management and configuration will be done by means of two related methods:

- JMX: for real-time minor management and configuration, such as setting non-critical, context sensitive parameters that change often. It may also involve temporarily stopping and starting services that need to be taken down for one reason or another.
- Inversion of Control pattern: for configuration of the overall system and configuration of rarely changing parameters.

It should be noted that under normal operation the use of JMX should be minimal and limited to the circumstances described. The primary means of configuration should be via an Inversion of Control Framework, as changes on an application level are “semi-hard” configurations that affect the actual structure and function of the applications.

In this section, when talking about service management & configuration, we are talking about configuring and managing data that is not subject to change often, and is related to the actual functioning of the architecture and application. For data, such as user readable texts etc, other custom means for configuration and management have to be used based on a case-by-case evaluation.

6.1.2 Statistics gathering and monitoring

In addition to the gathering of statistics for performance monitoring of the Branch Access Layer itself, the BAL will need to maintain statistics on a number of different business transaction outcomes. Some thresholds will be defined and where the statistics indicate that the threshold has been exceeded, appropriate alerts will need to be raised in real time.

Examples of the types of data to be collected and alerts to be raised are as follows:

- Authorisation agents and other online transactions: The transaction outcomes as seen by the counter application are recorded within the confirmation messages written at settlement time. These outcome codes will be recorded for individual Banks or other clients, when the percentage of failures over successes reaches a certain threshold within a defined time period for a specific client, an alert will be raised identifying the client.
- Settlement transactions from counters will contain the End to End elapsed time for the previous settlement transaction. The BAL will record the volume of transactions that fall within an array of response times. When the percentage of response times rises above a certain threshold within a defined time period, an alert will raised indicating that the system is in danger of missing SLT targets.

The set of statistics to be gathered and associated alerts will be agreed during the design phase.

6.2 Problem alerting

Problem alerting will be based on standard log levels. Errors that you would expect during runtime, such as erroneous input data will be logged at an intermediate log level, whereas more serious issues, such as database connection pools being exhausted will be logged at a critical level. Critical level issues will log to its own alerting framework that connects to the appropriate management/monitoring toolset.

To avoid duplicate issues around the same problem, some filtering logic will have to be applied to the logs; this will be done within the NetCool Object Server rather than within the OSR.

6.3 Statistics and performance reporting

Statistics and performance reporting will be done by means of JMX MBeans. These will be made available to the appropriate management and monitoring tools. The exact metrics to measure and points of measure will be decided on during design. More information on monitoring can be found in HNG-X BMX Monitor High Level Design (DES/APP/HLD/0015).

6.4 Start/stop of individual services

Individual Services will be made stoppable and startable by means of scripts that will be callable from the management tools. The OSR/BAL support guide will contain detailed information on these scripts and the way that they operate.

7 Security

The BAL is the front line of the data centre for business applications. It must provide a solid defence against unauthorised access, and log security related events. For branch Users, the BAL is responsible for user authentication, and provides role based access to user management capabilities.

Security is addressed in detail in section 2.5.5. This section intends to address peripheral areas and considerations:

- **Key management:** All management of cryptographic keys used by the BAL or supplied to the branch will be implemented by the Key Server. The design of the log on process described earlier within this document explicitly avoids the need for centrally managed keys that are specific to individual branches.
- **SSL Support:** Is assumed to be supported by the network layer rather than the Branch Access Layer.
- **Branch Access control and security enforcement:** Each service within the BAL is subject to Role Based Access control. The roles and permissions are stored in the BRDB. This is configurable on a per service level with certain services configured to not have access control.
- **Access from BAL to Branch database:** The BAL accesses the Branch Database as a specific user and will be granted a specific set of permissions. Use of this Oracle user will be restricted to the BAL processes. A separate user id is used for access to the training database.
- **Access from BAL to other online services:** The BAL initiates a TCP/IP connection to the online service components. There are no additional authentication mechanisms between the BAL and these components. Some services such as authorisation agents have additional authentication and replay protection mechanisms within individual transaction. As further protection of these interfaces, the routes between the BAL and these online services components must be protected by firewalls which will limit access to specific ports or URLs from sources other than the BAL.
- **Diagnostic / Trace logging:** The BAL must comply with any specific security requirement and not write sensitive data to diagnostic logs and trace files. In most cases we will avoid writing sensitive information to the logs by not logging business data. In the cases where we do need to log business data, such as when detailed diagnostics are required, we will implement a system that filters out the sensitive data from the business data before it is logged. It is expected that we can do this quite simply by adding sensitivity related flags to the metadata for business objects. This will allow the automated filtering of this data.
- **BAL Management Server:** The BAL management interface will be protected with password based access control, it is also planned that network firewall rules will restrict access to this service.
- **Non-functional requirements:** The SSL from the counter is terminated at the ACE blade rather than at the BAL to meet the requirement that encrypted data is not passed through firewalls. The BAL treats the network as open and therefore must protect itself against various network attacks including SQL injection type attacks.

8 Recovery and resilience

Recovery of a BAL server is very simple as the BAL architecture is stateless, any state stored is held in a database. Any recovery after failures such as corrupted files etc is best addressed through a full redeployment of the HNG-X applications.

It is also noted that backups are not required on BAL servers all server disks can be rebuilt using the automated deployment scripts. It may be possible that a snapshot type recovery method is useful for speeding up the recovery process, but this is a decision left to the data centre designers.

The physical resilience architecture is based on an *n+1* server's architecture. This means that a the loss of a single server is not disastrous; however it does mean that the maximum concurrency of the system will be impacted in proportion to the loss of hardware to total hardware. For instance, if we have 10 servers, and one fails, it will mean a loss of 10% of the capacity in terms of maximum concurrency and load.

At a finer grain level there will be multiple OSRs running on each server. If one of these fails it will simply restart. This will result in some small amount of transactions failing (timing out), and a small loss of data centre capacity, but otherwise there will be no impact. The OSR application lifecycle is managed by Tivoli. The Tivoli scripts will monitor OSR processes and automatically restart them upon failure or system restart. Tivoli monitoring will be notified of the restart of the OSR.

A BAL server must be able to start up and get to an available state with 2 minutes.

8.1 Start-up

On startup, the BAL will automatically start all of its configured services. This process will be managed by Tivoli using the provided startup scripts.

8.2 Branch database connection fail-over

The branch database connectivity, as detailed in section 2.2.5, will need to have intelligent routing, testing and fail-over capabilities for the BRDB connectivity. This means that the database connectivity will need to do the following things:

- Test the connection.
- If a connection fails to connect to a BRDB node altogether, change BRDB node.
- Handle total failure in a managed fashion. This process is described in detail in HNG-X Online Service Routing High Level Design (DES/APP/HLD/0050).

8.3 Online Service Routing fail-over

The Online Service Routing architecture will need to be able to handle failure of backend systems in a managed way. This means that it will need the following characteristics:

- Handle time-outs.
- Handle retries in the case of transient connectivity errors for both connections and messages..
- Handle alternative network routes.
- Handle complete failure. This will depend on the characteristics of the individual service in question, details will be provided in DES/APP/HLD/0050 for each service.

POL00140245

8.4 Disaster Recovery

As the servers are stateless, the fail-over and disaster recovery characteristics of the Blade Frames will be sufficient. These considerations are outside of the scope of the BAL. It is noted that the Blade Frames will provide significant facilities for failover and recovery.

As DR switchover is a manual decision so no prescriptive number of server faiures can be given. It depends on the load on the system and how well the remaining servers are coping. It is unlikely that say two OSR servers failing during heavy loading would cause a DR switch over. It is more likely that the remaining servers (how ever many that might be) start to show longer than acceptable response times would cause a switch over. If the remaining servers - however that many might be - can cope with the load and expected load before the "broken ones” are "fixed" then no switch over would be performed

9 Performance

9.1 Requirements and expectations

There are contracted levels of concurrency for the overall system and individual services as detailed in HNG-X System Qualities Architecture (ARC/PER/ARC/0001). These levels of concurrency must be met for all of the services. This implies that the concurrency must not only be able to be met, it also means that responses even under these high-volume times must be timely and reasonably fast.

The design limit for the system is 1150 TPS (Transactions Per Second), this is based on a 30% contingency for the overall 890 TPS concurrency identified in the Horizon system. Note that this is over all BAL instances, so if we deploy, for example, two instances on each of the 10 servers allocated, then the limit per instance is under 60 TPS, a much more achievable figure.

Long running transactions are a particular issue as each network transaction will allocate a certain amount of server resources during its lifetime. Queue management will be used to prioritise certain transactions over others which will also alter how resources are consumed. The tuning of these settings will be complex and will require large amounts of testing.

For example the contracted SLAs for network banking are 2.5 seconds (end-to-end), which has to be met under normal running. The BAL will only be allowed to use a very small amount of this time as most of it will be used in the authorisation agents and the counter. Details of the SLA requirements can be found in the system requirements.

Load balancing is to be implemented on a network level, and load will be distributed across up to 10 servers with up to 8 (but more likely 2 – 4) instances per server.

End of day reports represent a significant load on the BAL layer and will need to be considered carefully during performance testing.

9.2 Critical performance factors

The critical performance factors have for the most part already been evaluated and addressed in the architecture. It is however useful to highlight the most critical and likely points of potential performance bottlenecks, so that the project is explicitly aware of them throughout the project, from architecture through to development.

9.2.1 Performance of XML parsing and building

Looking at a scenario where we do not interact with any outside systems, XML parsing and building will most likely be the most time-critical portion of the architecture. The choice of method and framework with which to parse XML will be critical to performance and throughput of the system. Therefore the performance of this should be tested early on. This factor has been addressed in the initial recommendations on technology choices, and this will hopefully be sufficient, but the situation should nonetheless continue to be monitored throughout development.

9.2.2 Impact of authentication/authorisation model

The use of Public/Private keys in authentication, and in particular in authorisation of individual requests, needs to be monitored and measured. Initial tests using Apache XML Security on smaller documents during the architecture/technology evaluation phase pointed at the performance hit of validating a digital signature being miniscule (impact of 2ms per request). However, we need to keep an eye on this, as we get a system that resembles the end product more closely.

9.1.3 Performance of branch database

From experience, one can tell that the database is a common performance bottleneck in many J2EE applications. Correctly managed this should hopefully not be a problem, but the capabilities will be monitored during development, testing and rollout.

It should be noted that excessive and unnecessary database hits should be avoided at all costs. Wherever appropriate some measure of in-memory caching should be used, although due to the nature of the BAL architecture the opportunities for caching are limited due to all of the servers needing to maintain a consistent state. For instance we cannot cache session related information as session data must be always up to date across all servers.

Areas that are suitable for caching are the SQL Statements, FAD Hash maps, roles and permissions. Additional caches may be identified at design time but all caches will use a standardised caching mechanism. Because these data items change infrequently it is sufficient to have a manual process for flushing the caches (through the BMX).

9.1.4 Performance and throughput of third party systems

The most uncontrollable aspect of performance and reliability will be the interactions with third party systems and interfaces. The performance or even reliability of these systems is not something we have any control over whatsoever in the context of the HNG-X project. This implies that we cannot make any guarantees for the performance of services using these systems, apart from creating the best possible architecture for handling potentially unreliable connections, and connection disconnects/timeouts.

Third party system performance will be monitored by the BAL monitoring system though. This is important on its own, but is also essential for understanding the performance of the BAL itself.

The monitoring system will collect aggregated statistics of individual performance measures. More detail on this can be found in HNG-X BMX Monitor High Level Design (DES/APP/HLD/0015). The Queue based design of the OSR will help level out fluctuations in both demand and response times for third party systems. Timeouts will be used to ensure that excessive resource utilisation does not occur.

10 Migration

10.1 Overview

Because the BAL does not have an equivalent in Horizon there is nothing to migrate from. This means that there is little to do for migration, but there are still some considerations.

BAL services can be deployed at any stage before they are needed. When started up they will listen for incoming traffic with little impact on the rest of the data centre. Certain other services must be in place first. These include:

- The Key Management Server
- The BRDB
- Monitoring agents

A standard BAL will attempt to connect to the authorisation agents on startup. No transactions will be performed but some resources will be used in the agents to maintain these connections. Like all services on the BAL this is, of course, configurable, an OSR could be configured to not start up banking services if this was required. This gives us some flexibility in the migration process, for example, in the initial migration stages, the Banking Agents may not be running – e.g. if we activate PCI as the first service through the BAL for card payments only. It would be possible to do this before weekend B to get PCI out early.

A service that is not configured to start will not generate any system management events.

10.1.1 Horizon OSR

A specially configured OSR instance, which we have named the Horizon OSR, will be deployed alongside the standard OSRs. The Horizon OSR is configured to allow external card payment and Banking service (which has 3 separate clients CAPO, A&L and LINK) transactions from Horizon counters to be connected to the banking agents. The ETU service is not user by Horizon OSR This is to help achieve PCI compliance on the Horizon estate.

The primary need for the Horizon OSR is for DCS; it just happens that we have chosen to apply the PCI changes to NBS as well. If the PCI changes get brought forward into the Horizon solution then DCS will be the only use of this component.

As mentioned above, we may start up these services separately with the card payment service as the priority to get live for PCI.

The Horizon OSR does not include the standard authentication and authorisation filters as they are not supported by the Horizon counters. This means that the Horizon OSR port must be firewalled off from the HNG-X estate and the standard OSR port must be firewalled off from the Horizon estate.

Once the last Horizon counter has been removed from service the Horizon OSRs can be removed. It is expected that they will be replaced with an additional instance of the standard OSR (although listening on standard ports rather than Horizon OSR ports).

The Horizon OSR does not need access to the BRDB or the key management server. It will however, require monitoring to be in place.

10.1.2 Migration Report

The BAL servers will be required to supply the post migration report to the counters. The report data is generated by an overnight batch process. There is no special configuration required to perform this task; to the BAL it is just another standard report.

10.2 HNG-X Migration Enabling Upgrades for Data Centres

Note that the OSR-Horizon servers may be required to be built and in service before migration begins. This is not covered in this document though as the actual sequence of events has not been defined yet.

10.3 Data Centre Build

Deploy OSR servers, but do not start up.

Ensure Key Management Server is deployed.

Ensure BRDB is available and data is migrated/configured.

Ensure ACE blade is configured.

10.4 Move Wigan Network Management Servers

N/A

10.5 Data Centre Preparation

Start BAL servers and run tests.

10.6 Cutover Rehearsal

N/A

10.7 Migration of POL FS

N/A

10.8 Migration of Batch Services

N/A

10.9 HNG-X Specific Services

Start BAL servers, including standard OSR instances and Horizon-OSR instances

10.10 Migration of Online Services

N/A

10.11 Migration of Audit Services

N/A

10.12 Migration of Branch Services

N/A

10.13 Move Bootle Network Management Servers

N/A

10.14 Decommission Wigan and Bootle

N/A

10.15 Horizon Counter Changes for PCI Compliance

N/A

10.16 HNG-X Migration Enabling Upgrades for Counters

N/A

10.17 HNG-X Application Pilot and Rollout

No additional deployments, but performance must be monitored carefully as the load increases. It would also be useful to provide load projections based on measured data and planned migrations during this phase.

10.18 Branch Router Rollout

N/A

10.19 Counter Event Management Changes

N/A

10.20 Counter XP Upgrade

N/A (if it happens)

10.21 Post-Application ADSL Changes

N/A

10.22 Final Decommissioning

Remove Horizon OSR instances.

10.23 Estate Management Upgrade

N/A

11 Future Upgrades

11.1 Updating the version of Java

Java tends to be very backwards compatible, much effort goes into providing good backwards compatibility, quirks and all, so there should be no major problems.

One thing that an update does give the opportunity for is removing custom code from the application to leverage standard Java APIs that have been added. Long-term such a code-update will allow us to support less code, and our suppliers more.

In the update from 1.4 to 1.5 we have removed the custom threading code and replaced with standard code. We have also removed the IoC framework and replaced it with the open-source Spring Framework.

11.2 Migrating the Operating system

Provided the new underlying operating system supports Java, there should be no problems.

However it should be noted that the underlying socket characteristics of the operating system may impact the custom HTTP Multiplexer and the NIO Integration Framework performance.

11.3 Software Updates

From time to time the BAL servers will need software updates. This may be to add new features or to fix identified bugs.

Once a BAL release has been tested it can be deployed to the live estate. A safe deployment strategy can be planned without any service outage. This is achieved by updating the BAL servers one instance at a time. This update should be scheduled during a period where high server load is not expected, but apart from this can be done at any time. It is not necessary to schedule updates to be out of business hours, and it fact it may be better to schedule them during normal working hours as more people would be available for monitoring the update process.

It is possible to pilot feature updates in the live estate by deploying a new service handler alongside the existing service handler. The pilot service would have a unique name to differentiate it from the standard service. Specially configured counters could be set up to use the pilot service in preference to the standard service, while the rest of the estate would continue using the standard service. This is not how the data centre has been managed in the past but with HNG-X moving more business logic in to the data centre (compared with Horizon) this is something that could be implemented in future if required.

The BAL is packaged as a small set of packages that can be deployed independently. These packages are:

- The Java VM. The version may change from time to time, but this will be a standard java VM installation.
- The OSR. This includes compiled java classes and various configuration files. It is not expected that these configuration files will be edited in place, they will form part of the software release. Some configuration files will be created as a part of the installation process by scripts provided with the installation.
- A package for each service handler. This will include one java jar file and a configuration file. A service handler may require a particular release of the OSR. This will be documented in the release note for the service handler.

As deployment is within the data-centre on a fast network we do not need to be concerned with minimising deployment package size. It is more important for us to minimise the complexity and risk of deployment.

For this reason bug fixes and updates to the OSR (which may only affect a few files) are packaged up into a complete release that can be tested as a whole before deployment, it is not planned that we use patch files or other similar mechanisms as this adds unnecessary deployment complexity without providing any real benefit. Deployment will use standard HNG-X package deployment practices.

11.4 Adding or updating services

The BAL is designed to be as flexible as possible with regard to supporting new services. Certain classes of service can be added with only small configuration changes. Additional message types can be added to generic web services with no changes at all.

In some cases though the BAL will be required to perform new tasks that will require new code to be deployed. This is a simple task as all new code will be confined to specific modules. New service handlers can be deployed independently of existing service handlers, and new data centre interfaces can simply be added. These additions will require a new software release, something that is not a problem as there are a limited number of BAL servers and they all reside within the data centre.

While it is technically possible to deploy new modules within a running Java VM this is not something that we will be planning for. It is safer, easier, and more reliable to stop a BAL instance, reinstall, and restart as this results in the code always being in a known and tested state. This will not result in a service outage as each BAL instance can be updated individually leaving the other BAL servers available to service counter messages.

12 Testing and validation

Testing is formally described in more detail in the document HNG-X Testing Strategy (TST/GEN/STG/0001); however this section will detail different approaches and steps of testing appropriate for the BAL.

12.1 Unit testing

Unit testing of programming artefacts should be performed by developers during development (**not** after) with a framework such as JUnit.

A useful way of monitoring that this is actually being done is to have tools in the build system that periodically report the code coverage as a percentage of code that has associated unit tests.

Unit tests are an essential tool to achieve the following benefits:

- **A suite of automated regression tests**: unit tests allow developers to change and add to application code, and immediately verify whether they have broken anything previously working by running the suite of unit tests.
- **Protection against staff turnover**: Unit tests protect against dependence on an individual developer who may leave the company or go on vacation. Because code can be quickly regression tested, individual "ownership" of code can be better avoided, and other developers will be able to find the courage to fix, change or refactor code another developer has written.
- **Simplifies integration testing**: Integration testing of components and the application itself becomes easier, as most lower-level details will already have been tested to some degree.
- **"Living documentation" of a system**: The unit tests in themselves define the expected behaviour of the software, and thus becomes a living, always up-to-date document of requirements (if unit tests do not reflect requirements, they either will fail, or have to be changed).
- **Promotes decoupling and good development practices**: since unit tests should be run on a single class in isolation (potentially with mocked up dependencies), unit testing promotes good development practices, such as low coupling, few dependencies, defined object roles and development by interface rather than implementation.

Furthermore, when new bugs are found, a unit test should be written that tests for that explicit bug. Doing this will in effect give the project an effective regression test suite that protects against the recurrence of bugs that have already been fixed.

12.2 Code reviews

Recurring code reviews should be performed in groups every once in a while to assert that coding standards and practices are followed, and that the agreed upon architecture and design is adhered to (or changed as needed).

Code reviews are also a useful tool for sharing knowledge within a group of developers, and building a common view on how the problem domain should be solved.

Code reviews should occur every time a subsystem component is finished.

12.3 Service Interface validation testing

The interfaces of the Services will be defined as XML Schema Definitions. This means that inputs and outputs from a particular service will need to be tested. In the case of the Counter, its outputs need to be validated against the schemas of the services. This could be done by means of Schema validation in the BAL that can be turned on and off on individual services and depending on its operation mode.

When it comes to the Branch Access Layer, its responses will need to be validated against the response schemas available. In practice this means that an automated suite of tests using test inputs should be used to test different scenarios for each service. The responses for these test inputs should then be validated against the respective schemas of the services. It is important to get a wide number of possible response permutations out of the system, in order to test the different variations of possible XML responses.

12.4 Integration testing

By integration testing, I am referring to testing of individual components and services end to end (but not including the counter application), with all of their respective runtime dependencies tied together.

Integration testing can be automated and aided to some degree with the help of scripted tests in tools such as Apache JMeter, by running through different scenarios while asserting the expected results. Integration testing should, just as all types of testing, also test potential failure scenarios and see how the system copes with those.

It is planned to have a formal interface specification for BAL messages. This will be useful in testing for isolating the BAL from the counter.

12.5 Performance testing

Performance testing will test the performance characteristics of the system, and whether or not they are up to par. This should be done at the point where any major technology choices are tested/evaluated. Furthermore, performance testing can be done by using potential integration testing scripts created in automated test tools. For instance, the previously mentioned Jmeter is an excellent performance test tool, which can be linked together across a network of several computers to generate load.

A developer performance testing environment will be created to perform initial isolated performance testing. It is expect that a number of bottlenecks and issues will be discovered at this phase of testing and the results of the testing will result in software updates to address these issues.

A later stage of testing is planned that will include additional components, culminating in a test environment as close to the final environment as possible. We plan on bringing in Cisco ACE blade functionality to this environment as well as a small set of counter computers. We would also test on slow links by simulating slower network connections between the counter and the BAL.

12.6 Acceptance testing

Acceptance testing is the validation of whether or not the system as a whole fulfils the functional and non-functional requirements set out by the client.

This can be considered “whole system testing” in a sense, since all portions of the system, end-to-end will need to be tested.

Regression testing of acceptance tests can be automated to some degree by using scriptable and assertable test tools such as JMeter.

12.7 Build system and continuous integration

Many aspects of testing and validation can be automated through the build system. For instance unit tests and scripted integration, performance and acceptance tests can be added to the build process, meaning that you get an automated regression test suite run on each build of the software.

For further control and enforcement, continuous integration can be used with tools like *CruiseControl*: in effect this can be configured to run an automatic build with tests every 30-60 minutes with the latest code, meaning that any failures or broken builds will be caught early on, rather than at a later date when the problem can be harder to track down.

Furthermore, continuous integration allows us to plug in different tools that enforce policies such as unit testing, for instance tools that report on broken builds, unit test code coverage etc can be plugged in.

12.8 Other testing considerations

In addition to the mentioned testing areas and considerations, the system will also have to support various testing and training modes, this means that certain functions, such as branch session management must be startable and stoppable dynamically to allow for this type of testing in which certain portions of the system are shut down, so as to allow a certain level of isolation in end-to-end testing. We may also want to be able to retain functionality, but suppress any errors. One example of this would be having authorisation turned on, but suppressing authorisation exceptions and letting requests through, even in case of failure.

This also means suppressing some types of errors and allowing replay security to be shut down.

The way to achieve this is similar to starting and stopping individual services: it is by exposing the appropriate components as JMX MBeans that are manageable from the environments management and maintenance tools.

These testing considerations will be considered at HLD level on a case-by-case basis.

Any facilities added to make testing possible or more convenient must not compromise the security of the live system. This can be achieved by (a) logging clearly when in such modes and (b) implementing specific constraints that would show up in Live very quickly (e.g. restricted list of branches or UserIds).

13 Risks and assumptions

13.1 Assumptions

13.2 Risks

14 Requirements Traceability

Traceability of Business, Customer Service and System Requirements is detailed in HNG-X Testing Strategy (ARC/APP/RTM/0004).

A CPs included in this document

CT / CWP or CCN ref	FS CP ref	Description	Release	Commercial impact?
CCN 1214	HNG-X CP0016 (CP4382)	Multiple session Processing	HNG-X	Y
CCN 1202	(CP4305)	PCI Compliance	HNG-X	Y
N/A	HNG-X CP0136	Removal of Interstage from BAL	HNG-X	Y
N/A	HNG-X CP4747	Compression of messages between counter and BAL	HNG-X	Y
N/A	HNG-C CP0299	XML Message Compression	HNG-X	Y