UNIT 3 CASE STUDY

Structure **Page Nos.**

3.0 Introduction 51
3.1 Solution Overview 52
3.2 Solution Architecture View 53
3.3 Presentation Layer 53
3.4 Business Process Layer 56
3.5 Enterprise Application Integration Layer 57
3.6 High Level Functional Architecture 58
3.7 Presentation Layer Packages 59
3.8 Business Process Layer Packages 60
3.9 Business Function Layer Packages 60
3.10 Specific Solution Usability Related Elements 61
3.11 Summary 62

3.0 INTRODUCTION

The XYZ Bank has embarked on a significant strategic programme, which aims to replace major parts of the Bank's technology. The existing infrastructure has been designed as 'stove-pipes' for each delivery channel, which is restricting the flexibility of on-going developments. The Bank has now reached the position, where it cannot sustain further developments, which are required in support of the Bank's business strategy particularly with regards to the growth of, and interaction with, customers, products and channels. Most of the current applications are running on legacy applications.

The XYZ bank is looking for a solution, which will establish a platform that will allow longer-term technology implications. The scope of the solution covers the existing business functionality that is available within the Internet Banking System, and the bank's teller application that is available within the Branch Network. The scope of the project is restricted to 'migrate the existing functionality' onto a new infrastructure and architecture, and does not allow the introduction of any new business functionality.

The key business drivers are:

- To deliver an effective platform for multi-channel access to Personal Banking information processing services that will support effective integration of the Bank.
- To deliver an infrastructure that has a lifespan and lifecycle in line with the Bank's strategic requirements and that, which reflects the strategic nature of the investment.
- To provide improved capacity for any future developments of browser-based, Personal banking information processing services.
- To provide an environment that facilitates the introduction of the Euro currency in terms of identifying the particular currency that is tendered.
- To make use of acknowledged industry standards, technologies and best practices.
- To have regard for the overall cost of ownership of the solution, covering both development costs and ongoing lifetime costs.
- There is no scope for significant core systems (mainframe) re-engineering work to be undertaken during the lifecycle of the project.

3.1 SOLUTION OVERVIEW

The solution provides an 'Integration Layer' that enables multi-channel access from the Bank's browser-based delivery channels, both internal and external, to its main Personal Banking information processing services, primarily running on the Banks existing mainframe environment. It supports account applications and account transaction services for customer and internal staff, accessed over the Internet or via the bank's intranet. The solution is designed to run on a completely new set of mid-tier devices, using the IBM WebSphere product stack running on IBM's AIX operating system, and is based entirely on J2EE development technologies.

Key Issues

- The need to provide a robust, flexible and future proof (ten years) mechanism for internal and external multi-channel access to Bank Account Applications and Transactions.
- Addressing this issue would be the foundation of the architecture and design of the solution.
- The software architecture has been designed to support this, but may be limited by incompatibilities in the infrastructure, information and functional architectures.
- The need to adopt new working methods and skills in support of component-based iterative development.
- The team have been trained and have received external consultancy in the use of new methods, techniques and tools.
- The need to deliver effective documentation for the project and the use of sub systems and components that may also be used in other solutions.
- Currently there is little design documentation apart from the source code and configuration scripts, although this has begun to be addressed.
- The need to re-use as much as possible of the existing systems.
- Whilst the multi-channel integration layer is based on new technology, the majority of the business function processing and information storage remains on the mainframe. In addition, both customers and Bank staff can access and use the solution via their existing workstations and network infrastructure.

Key Risks

- The previous technology has no future development path and has reducing levels of support.
- To be a addressed by extending the support contracts with the current provider to cover the expected time until the future iterations enable complete replacement of the old solution.
- The proposed set of products and configurations is relatively complex and untested.
- The complete set of skills needed to deliver the solution is in short supply within the industry.

3.2 SOLUTION ARCHITECTURE VIEW

In essence, the solution provides an 'Integration Layer' that enables multi-channel access from the Bank's delivery channels, both internal and external, to the main Personal Banking information processing services, primarily running on the existing Bank mainframe environment. It supports account applications and account transaction services to customers, as well as providing a mechanism for customers and internal staff to communicate in a secure manner.

The high-level software architecture (see *Figure 1*) is based on the bank's Software Layering Model, with the solution being constructed in a modular fashion and implementing discrete responsibilities for Presentation, Business Process, EAI and Business Function.

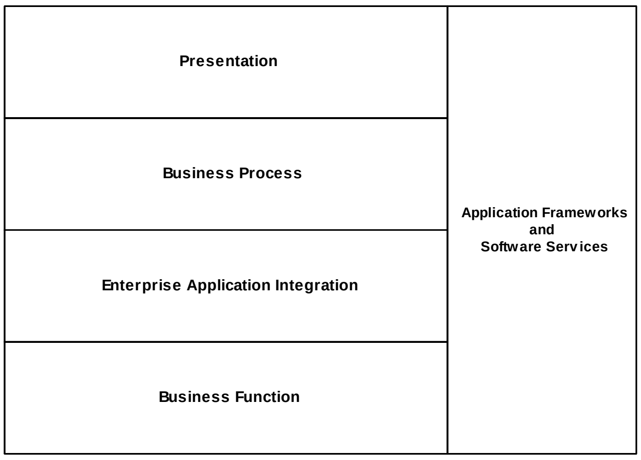
Presentation
Business Process
Enterprise Application Integration
Business Function
Application Frameworks and Software Services

Figure 1: Software layering diagram

The Presentation layer only supports browser-based channels, via the delivery of HTML generated by Java Server Pages.

The term 'Integration Layer' loosely refers to the Business Process and EAI layers, which are based entirely on J2EE development technologies. Without exception, all the business processes that have been developed are 'single-shot' tasks and there has been no requirement for any long-running processes that would have necessitated some form of business process engine (workflow).

Most of the Business Function layer already existed on the mainframe, though it has been re-developed (where economically viable) to provide increased modularity and has been made available via a generic CICS transaction interface that calls specific business functions.

3.3 PRESENTATION LAYER

Figure 2 identifies the significant software components and indicates how these map to the high-level software packages (shown in grey) that are described within the bank's Software Layering Model.

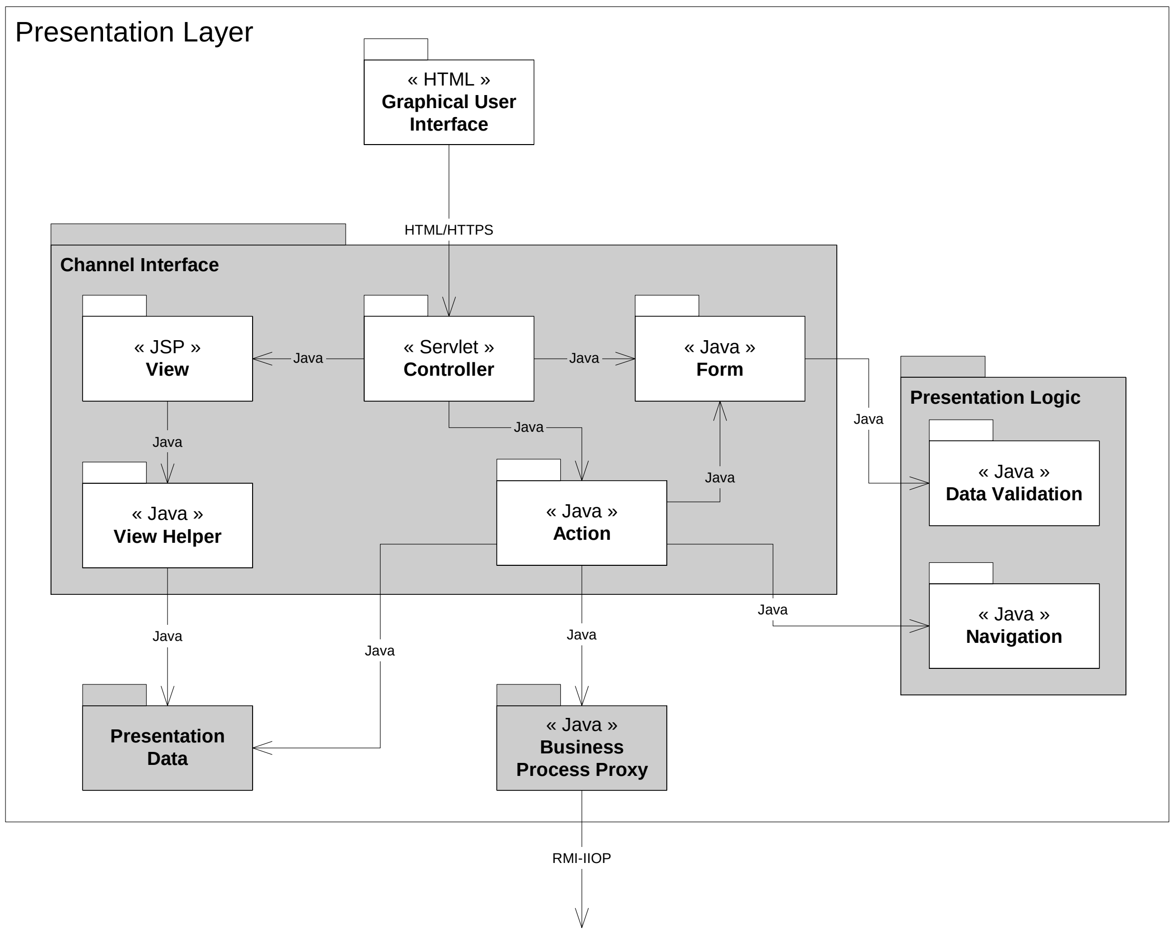

Figure 2: Presentation layer software context diagram

The Struts framework (from the open source Apache Software Foundation) should be used as the basis for the presentation layer. Struts utilises open standard technologies and encourages software architectures based on the Model-View-Controller (MVC) Model 2 design pattern. The framework was selected since it is one of the most widely adopted web presentation frameworks available off-the-shelf and if significantly reduces the development effort required on the project. As this is an open source framework there is a risk that a lack of a formal support contract from a supplier could lead to production issues if any significant bugs are found in the Struts code. This risk is considered acceptable since, the source code is available for update if required and the Java skills needed are prevalent on the project and will remain in the production support teams.

Controller

Controller components are responsible for receiving requests from the browser, deciding what business logic function is to be performed, and then delegating responsibility for producing the next stage of the user interface to an appropriate View component.

Within Struts, the primary component of the Controller is a command design pattern implemented as a servlet of class *ActionServlet*. This servlet is configured by defining a set of *ActionMappings*, which in turn define paths that are matched against the incoming requests and usually specifies the fully qualified class name of an Action component.

Form

Form components represent presentation layer state at a session or request level (not at a persistent level).

Within Struts, the Form components are implemented as *ActionForm* beans.

View

View components are responsible for rendering the particular user interface layout, acquiring data from the user and translating any user actions into events that can be handled by the Controller.

Within Struts, the View components are implemented as Java Server Pages (JPSs).
In addition to using the Struts framework, the Tiles framework (bundled as a set of extensions to the Struts code) is also used. The Tiles build on the "include" feature provided by the Java Server Pages specification to provide a full-featured, robust framework for assembling presentation pages from component parts. Each part ("Tile") can be re-used as often as needed throughout the application, which reduces the amount of mark-up that needs to be maintained and makes it easier to change the look and feel of the application.

View Helper

View Helper components encapsulate access to the business logic and data that is accessed and/or manipulated by the views. This facilitates component reuse and allows multiple views to leverage a common 'helper' to retrieve and adapt similar business data for presentation in multiple ways.

A discrete View Helper has not been implemented as such, however a number of custom tags and utility classes have been developed to support the storage and retrieval of 'customer' data from the HTTP session object.

Data Validation

Data Validation components are responsible for simple field-level validation of user input data, for example type checking and cross-validation of mandatory fields. These components are not responsible for any business related validation, which is handled accordingly within the lower layers.

Validation of any input data is achieved via the use of *ActionForm* beans, though any common validation logic is grouped within the Data Validation package.

Navigation

The Navigation package provides the logic to determine the most appropriate view to be displayed to the user.

Action

Action components act as a facade between the presentation layer and the underlying business logic.

Within Struts, this is achieved using Action classes that invoke specific business processes via the use of Web Service requests through the Business Process Proxy.

Presentation Data

Presentation Data components are responsible for maintaining any state that facilitates the successful operation of the presentation layer logic. For example, the Controller and View Manager must have some way of knowing where the user is within a process, as well as have somewhere to store transient information such as the state of the user session.

Specifically, user session state is maintained within the HTTP session object provided by the appropriate J2EE container. This approach is both mature and proven and conforms to one of the standard session handling techniques as detailed within the J2EE specification.

Session tracking is achieved via the use of non-persistent HTTP cookies, which is a mature, standard for J2EE session tracking.

Business Process Proxy

Business Process Proxy components should act as a local representation of the Business Process Gateway.

The Business Process Gateway itself will be implemented as a command design pattern using an EJB and the Business Process Proxy as Java classes that encapsulate the lookup and instantiation of the gateway, such that the gateway can be remotely located from the proxy.

3.4 BUSINESS PROCESS LAYER

Figure 3 identifies the significant software components and indicates how these map to the high-level software packages that are described within the bank's Software Layering Model.

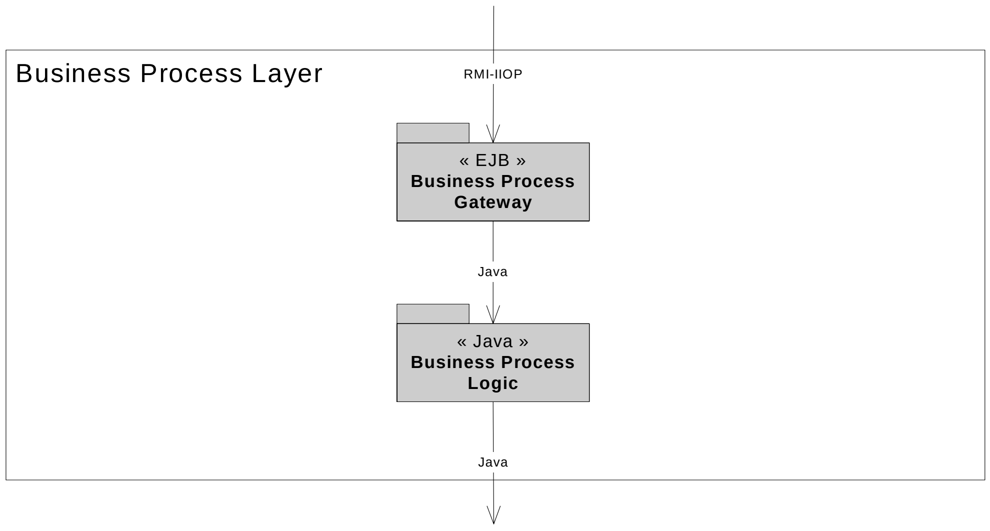

Figure 3: Business process layer software context diagram

Business Process Gateway

Business Process Gateway components will act as a wrapper to the underlying business process logic and translate simple requests into appropriate calls to execute specific business logic. The 'gateway' ensures that a generic interface is offered to clients and insulates clients from the complexities of the specific Business Process Logic interface.

The Business Process Gateway will be implemented as a command design pattern, using a single EJB with a single 'execute' method. The EJB utilises a configuration file that maps each specific request to the appropriate business process logic class and method for execution.

Business Process Logic

Business Process Logic components will implement the flow of work associated with requests for business logic execution, which could be anything from a simple single task to a complex combination of activities and tasks invoked over long periods and possibly involving many different resources.

There is no requirement to manage process or activity state across multiple tasks (workflow). N.B. An example, exception to this is, the 'logon' activity that invokes multiple tasks, albeit without the need to manage state beyond the boundary of the activity.

The Business Process Logic components will be implemented as Java classes that will inherit from the *AbstractBusinessProcess* class and will implement the *BusinessProcessGatewayInterface*. The classes will map directly to the modelled use cases, with any user interaction (to capture input data for a task) being handled by the presentation layer and the task only being invoked once all the required input data is present. There is then no further interaction with the task until the resulting data is returned to the presentation layer once the task has been completed.

3.5 ENTERPRISE APPLICATION INTEGRATION LAYER

Figure 4 identifies the significant of software components and indicates how these map to the high-level software packages that are described within the bank's Software Layering Model.

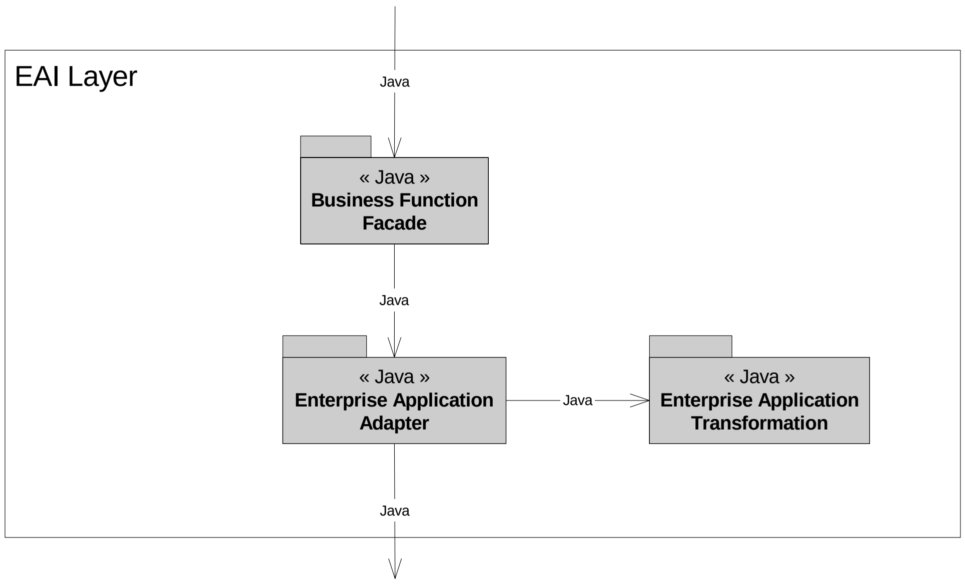

Figure 4: EAI layer software context diagram

Business Function Façade

Business Function Façade components will provide a unified interface to the set of business function interfaces that will be implemented in the underlying enterprise applications, thus enforcing a weak coupling between the façade's clients and the underlying applications.

The façade will be implemented as a single generic interface in the form of a Java class (*Busi*) with a single 'send' method that accepts three parameters, namely the relevant data, the associated context and a string that identifies the particular function to be invoked. Java reflection and configuration files will be then used to determine the particular external resource to communicate with, the format of the message and the expected return types. This implementation, though quite complex, is highly configurable and ensures a weak coupling between the business process layer and the underlying applications.

Enterprise Application Adapter

Enterprise Application Adapter components will provide transport mechanisms to forward requests for business function onto the underlying enterprise applications that implement the requested functions. This decouples the Business Function Façade from the physical implementation of the Business Function, which may be local or remote.

The Enterprise Application Adapter components will be implemented as stateless session EJBs, with a combination of XML configuration files, JNDI, and Java reflection being used to determine and instantiate the particular external resource to communicate with.

Four specific ‘adapters’ should have been implemented, namely:

- *HostAdapter* to communicate with the mainframe-based retail banking system via the Java Message Service (JMS)
- *DatabaseAdapter* to communicate with the mid-tier DB2 system to via JDBC
- *JavaMailAdapter* to communicate with Message Transfer Agents (MTAs) that support the Simple Message Transfer Protocol (SMTP), via the JavaMail interface
- *DummyHostAdapter* to communicate with a local ‘test harness’ that mimics the mainframe-based retail banking system when it is not available

Enterprise Application Transformation

Enterprise Application Transformation components will be responsible for transforming messages from one format to another, including translating objects from one development language to another where applicable. Typically such transformations will occur between the EAI layer’s native format (Java) and the format supported by the particular underlying enterprise application. These transformations are usually bi-directional, occurring once in either direction.

A comprehensive transformation mechanism has been implemented, based on configurable Templates, to support the following:

- Transforming the requests for business function into the specific message formats expected by the mainframe-based retail banking system and other external resources.

- Transforming specific field types from one format to another, e.g. a Java date field to a Cobol date field.

- Encoding and decoding sensitive data before and after transmission, e.g. the customer PIN and SPI data.

3.6 HIGH LEVEL FUNCTIONAL ARCHITECTURE

The Solution’s functional architecture (see *Figure 5)* has been defined to provide a potential set of groupings that could be of benefit in the future, because much of the business function processing is already implemented within the Bank’s mainframe that, has been re-used with limited changes. As a result this iteration of the project has not created namespaces to reflect this potential set of groupings.

However, the following section does identify this grouping and then uses it to group the specific business process tasks and business function operations implemented to show how we would have been able to start structuring meaningful business units of work to understand and change the systems base easily.

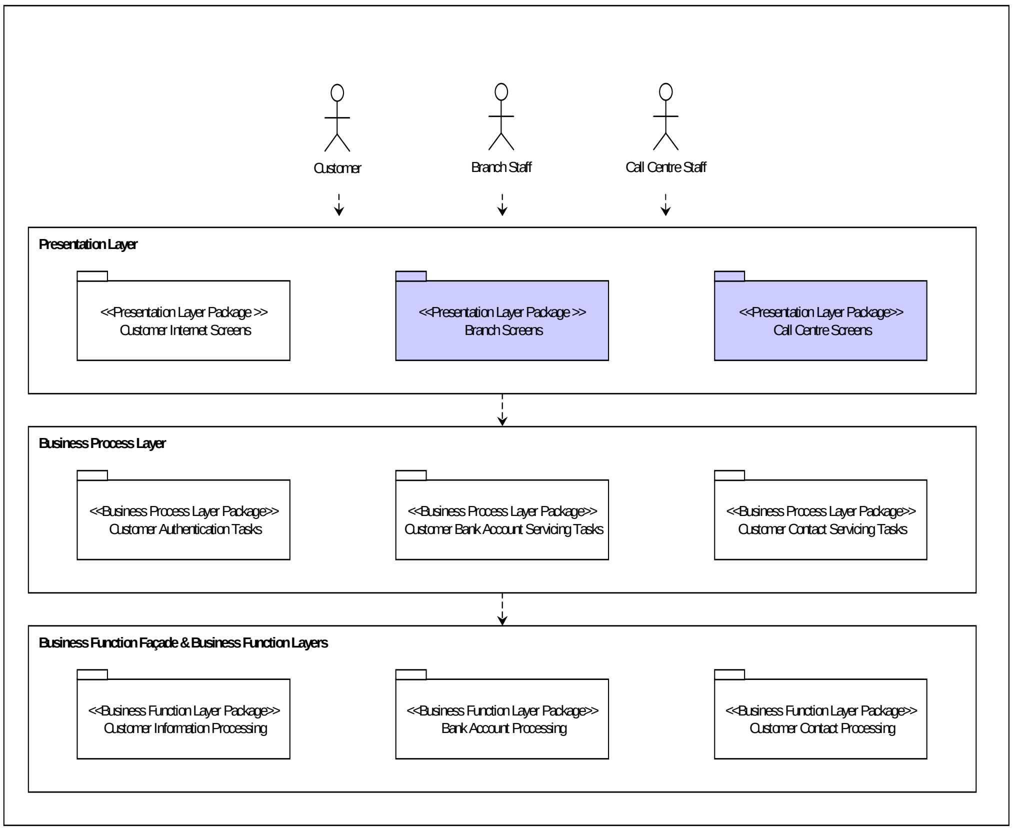
Customer
Branch Staff
Call Centre Staff
Presentation Layer
<<Presentation Layer Package >> Customer Internet Screens
<<Presentation Layer Package >> Branch Screens
<<Presentation Layer Package>> Call Centre Screens
Business Process Layer
<<Business Process Layer Package>> Customer Authentication Tasks
<<Business Process Layer Package>> Customer Bank Account Servicing Tasks
<<Business Process Layer Package>> Customer Contact Servicing Tasks
Business Function Façade & Business Function Layers
<<Business Function Layer Package>> Customer Information Processing
<<Business Function Layer Package>> Bank Account Processing
<<Business Function Layer Package>> Customer Contact Processing

Figure 5:The Solution's Functional Architecture diagram

3.7 PRESENTATION LAYER PACKAGES

The solution was originally focused on delivering screens that could support multiple channels and multiple users. This genericity was quickly relaxed for the different classes of users (customer, branch and call centre) because of the potential benefits of interface optimisation. But the intention remains to re-use as much as possible across each of these user groups.

Within the customer Internet screens package there is additional variation that has been introduced by the need to provide different look and feel for branch and internet banking. This iteration has therefore, focused on the customer Internet screens package and its initial support for internet banking processing.

The presentation layer is logically partitioned into three separate packages of screens:

Customer Internet Screens Package

This package provides the screen interfaces needed for customer Internet access to the Bank Account and Customer Contact Servicing business process.

Branch Screens

This package will provide the screen interfaces needed for branch staff access to the Bank Account and Customer Contact Servicing business process.

Call Centre Screens

This package will provide the screen interfaces needed for call centre staff access to the Bank Account and Customer Contact Servicing business process. The call centre package may be addressed within the project or a separate Call Centre Integration project depending on their development over the coming months and years. A small part of this package has been addressed within this iteration of the project that has delivered the Operator Application for managing secure e-mails.

3.8 BUSINESS PROCESS LAYER PACKAGES

A major objective of the project is to begin to share business processes across different interaction channels and communities of users, where appropriate. The business processes have been implemented as single shot tasks with no processing provided for the creation and state maintenance of long running processes as there was no requirement to manage process or activity state across multiple tasks. Future extension of the definition and management of these processes may require a more expansive approach to process definition and management to be developed.

The set of tasks undertaken, within each business process layer package, are described below. Within this iteration the business process tasks were designed to replicate much of the existing functionality (available using the old technology) rather than to add new capabilities. The business process layer is logically partitioned into three separate packages of business process tasks:

Customer Authentication Tasks

The customer authentication task package implements the following tasks:

- Customer Logon and View Accounts
- Customer Logon, View Accounts and Secure Messages
- Customer Logoff

Customer Bank Account Servicing Tasks

The customer bank account servicing task package implements the following tasks:

- Customer Bank Account Servicing Requests
- Manage Bill Payments
- Manage Funds Transfer
- Manage Customer Requests
- Manage Standing Orders
- Manage Statements

Customer Contact Servicing Tasks

The customer contact servicing task package implements the following tasks:

- Manage Secure Messages

3.9 BUSINESS FUNCTION LAYER PACKAGES

Within the project, the aim was to reuse business function processing on the mainframe providing façade processing (to access the mainframe processing) in the middle tier and adding new business functions in the middle tier for processing. Within this section, the business function operations packages represent both the business function façades and the business function implementations. The business function layer is logically partitioned into three separate packages of business function operations:

Customer Information Processing

The customer information-processing package implements the following business function operations:

- Customer Enquiry
- Customer Processing

Bank Account Processing

The bank account processing package implements the following business function operations:

- Account Enquiry
- Account Processing
- Customer Request Enquiry
- Customer Request Processing
- Funds Transfer Enquiry
- Funds Transfer Processing
- Bill Payments Enquiry
- Bill Payments Processing
- Statement Service
- Standing Order Enquiry
- Standing Order Processing

3.10 SPECIFIC SOLUTION USABILITY RELATED ELEMENTS

The main elements of the solution that support usability are the following:

- screen designs,
- task dialogue design,
- responsiveness and controls built into the mainframe implementation of business function, and
- conformance to the DDA.

The first three elements described above were developed in response to the specific behavioural and performance needs identified by the use cases that described the user interaction with the system. Each use case involves the interaction with a system interface (such as a screen), the invocation of a business task and the execution of a business function. The solution has been designed to enable each use case to provide usable interaction for the solution users.

Learnability

The solution needs to be easy to learn such that an external user can effectively use the different channels available when they first use the system. The solution should be designed to be easy to learn. It is intuitive to the degree that someone using it for the first time can find his/her way around the system with little or no assistance. Help screens are provided to assist the users.

Likeability

The different Bank channels of interaction with customers pride themselves on enhancing the customer's view of the Bank and its commitment to people and ethics. The solution would have to be designed to be "attractive" to customers and promote a positive view of the Bank.

Productivity

Customer and staff time is important and the solution needs to assist improvements in productivity. The solution should be designed to minimise the time and effort needed to carry out the tasks enabled by the system. This was somewhat compromised by the decision to move all screen processing to the server to meet other requirements. However, the creation of an effective user interface was managed by early prototyping with the system users to agree on the best solution for the interface requirements.

3.11 SUMMARY

The solution is prescribed along modular lines. It provides both vertical and horizontal scalability. The software should be developed as components with clear responsibilities servicing each of the application architecture layers. The components and objects within these layers are carefully designed following the principles of loose coupling, cohesion and clear management of pre and post conditions.

Components managing the interfaces between each application architecture layer should provide clearly defined APIs to enable the flexible combination of functionality offered by each layer, and localisation of the impact of change to those components. This provides the basis for managed extendibility of the solution.