WebSphere® IBM WebSphere Multichannel Bank Transformation Toolkit

Version 6.1

Solution Architecture

Note!
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 35.

This edition applies to Version 6, Release 1, Modification 0, of *IBM WebSphere MultiChannel Bank Transformation Toolkit* (5724-H82) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. You can send to the following address:

IBM China Software Development Lab
Bank Transformation Toolkit Product
Diamond Building, ZhongGuanCun Software Park, Dongbeiwang West Road No.8,
ShangDi, Haidian District, Beijing 100193 P. R. China

Include the title and order number of this book, and the page number or topic related to your comment.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Solution architecture overview

This document is mainly for Solution Architects, who require an overall description of what the IBM® WebSphere® Multichannel Bank Transformation Toolkit (Bank Transformation Toolkit) provides and how it may be used to build a solution. This document is also useful for IT professionals and executives who require a broad understanding of the architecture of this product and the strategy for its implementation.

Readers of this document are assumed to be familiar with object-oriented software and related development techniques, and to have a general knowledge of J2EE and related technologies, network computing, and Internet technologies.

Introduction

The IBM WebSphere Multichannel Bank Transformation Toolkit is a component-based toolkit for developing enterprise e-business applications. The Bank Transformation Toolkit enables the development of interfaces to the services of a financial institution's information system so that they become ubiquitous through all delivery channels (such as the traditional branch, call center, banking kiosk, Internet banking, and mobile access). This minimizes the need for developing new code and reduces the time required to make new financial services available to all delivery channels.

The architecture and technological approach of the Bank Transformation Toolkit creates retail delivery solutions that preserve investment in existing enterprise systems while accounting for the inherent instability of any infrastructure due to innovations that appear frequently in the high-tech industry. While providing a way to preserve existing systems, the Bank Transformation Toolkit is not tied to one particular platform because it is built on Java™, the programming language of choice for handling platform change. The toolkit also takes advantage of existing platforms and technologies such as Eclipse, Web Services, J2EE, Struts, and so on. The toolkit runtime architecture is based on the J2EE architecture with extensions, and many development tools the toolkit provides are Eclipse plug-ins.

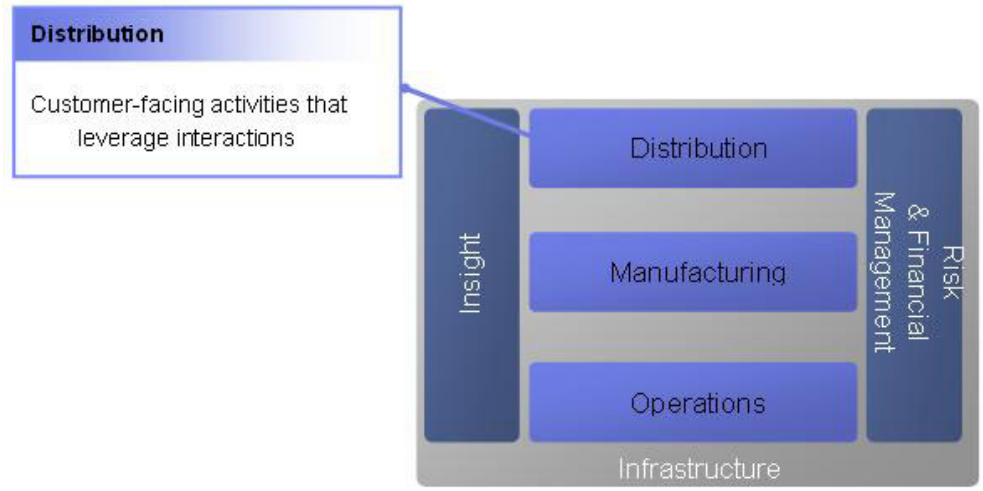
The Value of Bank Transformation Toolkit to Business

This section lists the value of Bank Transformation Toolkit to your business:

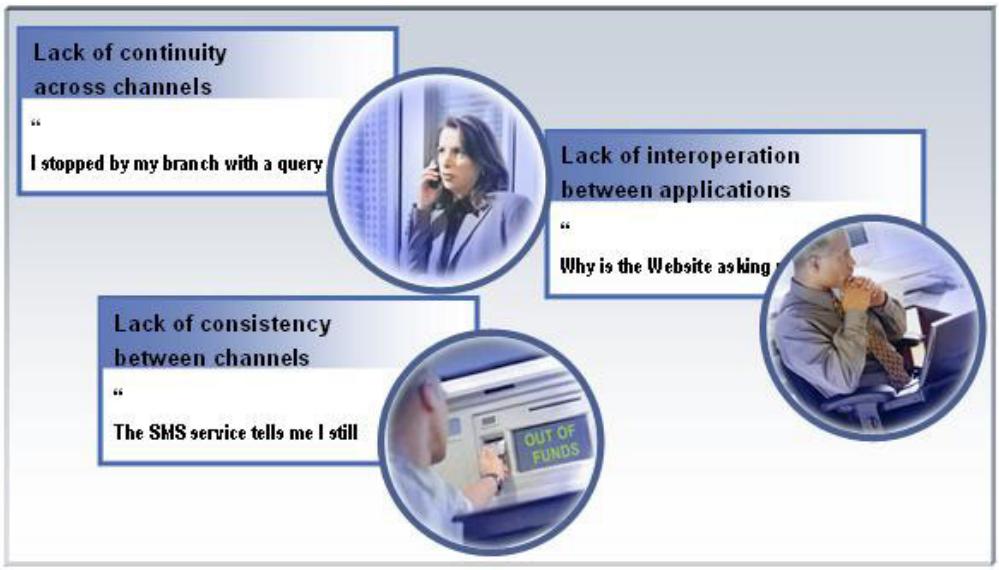
The importance of Channels to the Banks

Financial institutions compete and excel across six common competencies: Distribution, Manufactory, Operations, Insight, Risk and Financial Management. The institutions are striving to deliver them faster, at lower cost, and with higher quality than the competitors. Channel applications are the IT systems that facilitate the Distribution.

The following diagram shows the six competencies:



A lack of channel integration can cause customer dissatisfaction as shown in the following figure:



For this and other reasons, banks are investigating the channel applications. Other reasons include the following:

- Increased domestic and foreign competition
- Increased choices and ease of switching
- Customer service is improving in other industries
- Increasing multichannel contacts are changing expectations
- Constant innovation and improvement
- The branch is the hub of most banking activities and the most visible distribution outlet
- The online experience is essential to managing the relationship and researching new opportunities

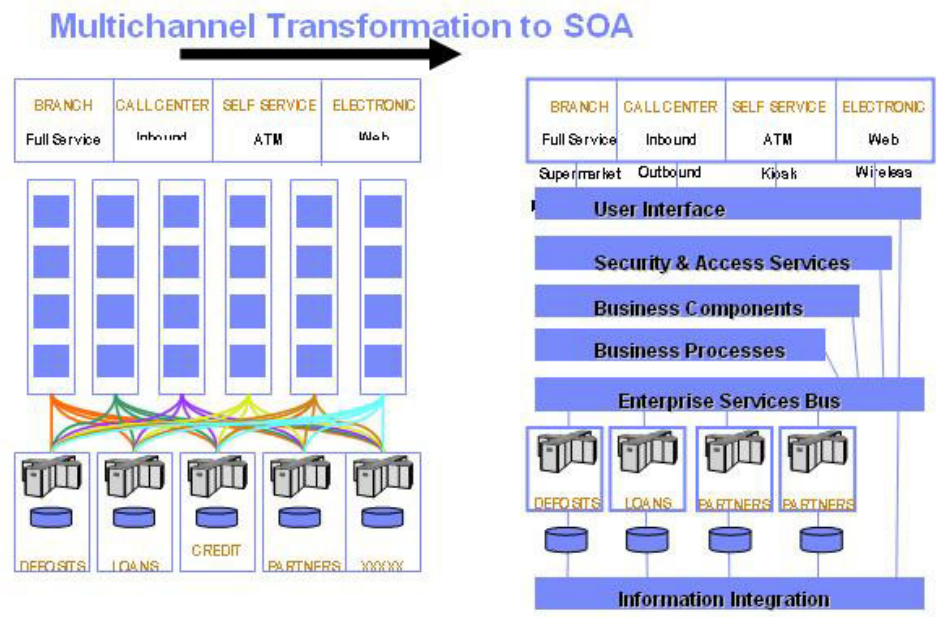
Bank channel applications enable nearly all the customer interactions.

The ability to quickly enhance channel applications, while keeping them integrated and consistent can be the source of competitive advantage.

Multichannel transformation

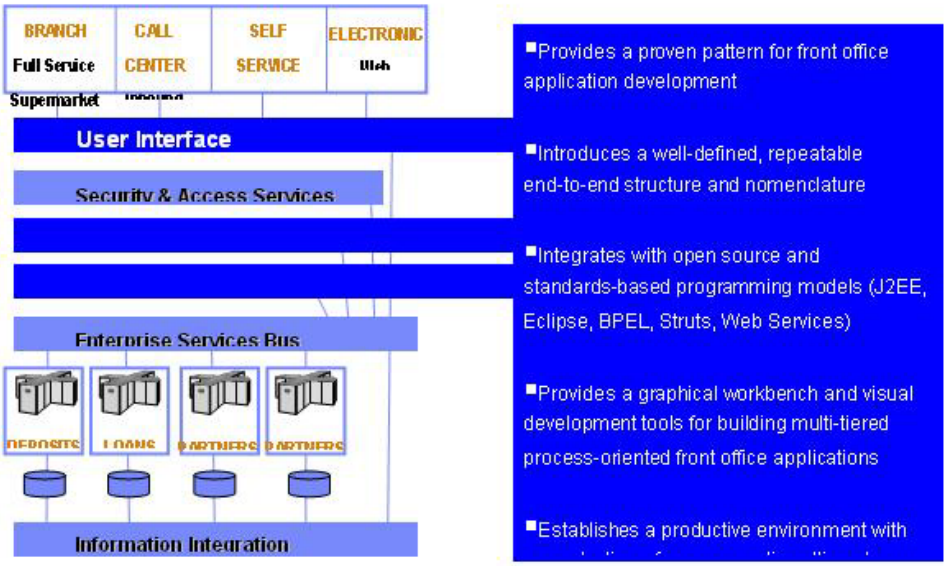
To pave the way for effective channel application investment, IBM provides a multichannel transformation based on a Service Oriented Architecture (SOA).

The following figure shows the multichannel transformation to SOA.



A key component of the Multichannel architecture that IBM provides is WebSphere Multichannel Bank Transformation Toolkit (BTT).

BTT provides a common framework for building integrated and consistent channel applications as shown in the following figure:



How BTT helps

Bank Transformation Toolkit can help increase and ensure the benefits of IT investments in channel applications:

- Channel applications are built on a common framework, leveraging new capabilities across all channels:
  - Home Banking
  - Teller
  - ATM
  - Contact Center

  By leveraging investments across all channels (as opposed to requiring duplicate investment), projects are completed with less effort and with greater benefit to the business.
- Consistent and seamless user experience, including multiple countries with separate core systems
- Simple, agile and predictable development process leading to:
  - More predictable success in development projects
  - High confidence in IT
  - High level of flexibility in obtaining and reallocating resources
- Proven and stable runtime environment:
  - Less risk when deploying new releases
- A smooth migration process for each release of BTT:
  - Ensure that investments in IT are leveraged over time
  - Existing systems can benefit from continuous new BTT features

Architectural objectives

The architectural objectives of the IBM WebSphere Multichannel Bank Transformation Toolkit align with IT strategies that have a basis in controlling costs over time. Following are the objectives:

- **Reduce costs** - A network computing architecture should exploit the network in order to reduce costs. It allows reduction of the computing resources required on

the client and supports deployment on network computers, using the network as a vehicle for on-demand distribution of software components. In addition, the architecture supports deployment of reusable business components in a managed server environment.

- **Preserve investment** - An important goal is to preserve the financial institution's investment in host systems and computing infrastructure, as well as in the toolkit-based solutions themselves and other new technologies. This makes it important to carefully consider technology selections in order to ensure that they are strategic and will have enduring value.
- **Offer choices** - Allow customers the flexibility to choose their hardware, operating systems, networking systems, databases, communication protocols, and third-party software products. The system must also support flexible distribution of function and data based on the network environment and physical topology.
- **Evolve gracefully** - The system must be flexible and resilient to both business and technological changes. This helps to support rapid application development and to increase competitiveness by improving time to market.
- **Provide manageability** - Once deployed and in production, the system must be easy to manage and resilient to changes in the runtime environment.
- **Allow incremental investment** - The system must support the ability to incrementally develop and deploy new business function and technology. In addition, it must support the ability to include new toolkit-based solutions as they become available.
- **Maximize usability** - The system as a whole must be well suited to the needs of its users: not only end users but also developers and systems management personnel.
- **Maximize reusability** - The system must be constructed in such a way as to maximize reuse of components in all retail delivery solutions. In addition, it must be able to meet the diverse needs of solutions and access channels in financial institutions around the world.

Architectural principles

The architecture must be open, scalable, and easy to implement. These principles are related to the architecture objectives, and are the basis for the platform selections, programming model specifications, and overall non-functional requirements of all the toolkit-based solutions. The major architectural principles of open, scalable, and easy to implement, presented below, demonstrate how the IBM approach for building robust, cost-effective enterprise systems support the architectural objectives. Following are the principles supported by the Bank Transformation Toolkit:

- **Open**
  - **Supports industry standards** - The architecture is open because it uses open industry and e-business standards such as TCP/IP, HTML, HTTP, J2EE (Java, Java Server Pages, JCA, JDBC, EJB, and so on) and Web Services wherever possible. These standards provide a solid foundation and make it easier to use available proven components instead of building custom ones, and to change vendors and implementations to satisfy changing business requirements. Industry standards tend to be strategic and have longer life spans because of the high levels of investment and commitment involved with creating them.
  - **Is extendable and customizable** - The toolkit is extendable and customizable at many different layers within the architecture. This means it can be used in

a wide range of situations and can accommodate specialized requirements that are specific to an individual customer, country, or region.

- **Provides insulation** - The toolkit isolates and abstracts interactions with other systems to insulate toolkit-based applications from the specifics of other systems. In a global solution, this is essential to provide the flexibility to adapt to many diverse environments, particularly different host systems and databases. The programming model of the toolkit insulates applications from changes in the underlying technology.
- **Preserves investment** - The principles listed above ensure the preservation of customer investments. The toolkit safely preserves the investments in current hardware, software, operating systems, network, communication infrastructure and protocols, and back-end subsystems of the customer environment.

- **Scalable**
  - **Supports three logical tiers** - The benefits of a logical three-tier architecture such as the network computing architecture are well known. The network computing architecture is logical in that it specifies that the presentation layer must be decoupled from the business logic, which must be decoupled from the data access layer, but it does not specify how to physically deploy the tiers. Although this approach is a form of isolation, it also provides scalability by allowing each of these layers of the system to change independently of the others. That is, the platform selections and design of each layer can change without impacting the rest of the system. This architecture also requires that the presentation layer be ″thin″ to realize the goals of network computing. This means that workstations with a small amount of physical memory and no virtual memory can download and execute the application. The main objective of the solution architecture is to support the model of a multiple-tier network computing application while also allowing engagement teams to implement solutions based on other application models such as a two-tier ″fat client″ application.
  - **Supports replaceable components** - Components are packages of system function with established interfaces and a predetermined execution environment. As long as a component is within its required execution environment and it interacts with other system components through its public interfaces, it is replaceable with minimal effort. This construction enables high levels of reuse and allows the system to evolve without causing large ripple effects. It also allows the implementation of components and their execution environments to vary to meet performance or scalability requirements.
  - **Provides enterprise topology independence** - This notion extends the idea of a logical three-tier architecture so that not only are the three tiers independent of physical location, but system components are independent of any specific physical topology. This makes toolkit-based solutions highly flexible for deployment in different environments by allowing customers to configure the system as needed to achieve the scalability desired for their environment.
- **Easy to implement**
  - **Uses visual programming** - Where possible, toolkit-based solutions use visual programming to assemble the application from parts. This technique is particularly effective in developing application screens and rapid assembly of graphical user interfaces.
  - **Separates analysis from design** - Analysis should be a separate process from design and has its own distinct work products. Solutions of this product suite should use analysis to form an entirely logical representation of system

function that is independent of technology or implementation. This helps to retain the value of earlier development effort even if the implementation must change entirely.

- **Provides a development methodology** - This solution provides a methodology for guiding the development process in an engagement project to make solution implementation easier and the deployment faster.
- **Is transaction-oriented** - Most projects require a solution in which an enterprise-centric back-end system executes most of the application business logic and the front-end of the solution, running in a delivery channel, must behave as a transaction posting engine to run the transactions in the back-end system. The Bank Transformation Toolkit excels at this type of solution and optimizes the processing of the transactions especially in high transaction volume environments.
- **Minimizes development effort** - The toolkit highly promotes the externalization of parameters so that business operations behave differently depending on their specific set of parameters. This enables solutions to delivery new functionality without requiring new code, simply by adding new external parameters to the system. One example is the toolkit business processes that are defined with BPEL. This enables toolkit application developers to edit process logic using visual design and modeling tools.

Multichannel consideration

The IBM WebSphere Multichannel Bank Transformation Toolkit provides an architecture for building applications that are deliverable on multiple channels. Enterprises within the banking and financial services industries have successfully deployed the toolkit in various topologies as the infrastructure for enterprise systems with high transaction volumes. While the following topologies are specific to the banking and financial services industries, for which the toolkit was originally conceived, the ability of the toolkit to handle multiple business distribution channels is generic and can apply to other industries.

Bank teller

A bank teller application topology consists of a number of client workstations with financial devices attached. The workstation downloads the client application on request from a Web server. The client applications, which mainly deal with presentation and local financial device handling, have access to the branch server (that is, the solution application server) using the HTTP or SSL protocols.

The solution application server provides common services such as electronic journaling and parameter tables to the client workstations, as well as access to the transactional logic of the back-end enterprise servers. A toolkit server application can also be deployed on the physical server for a regional or central data center without changes to the application.

Internet banking

In an Internet banking topology, users obtain access to financial services through a Web browser (or other device) connected to the Internet. The user interface is normally HTML with additional technologies such as JavaScript™, DHTML, or XML. In such an environment, the solution application server is able to process requests from Web browsers (or other devices that issue HTTP requests), obtain the proper data from enterprise servers, and generate the appropriate view for the client device to display, using HTML pages for Web browsers or XML messages for those devices that support it. The application server is usually located at the central site, and is protected by a firewall.

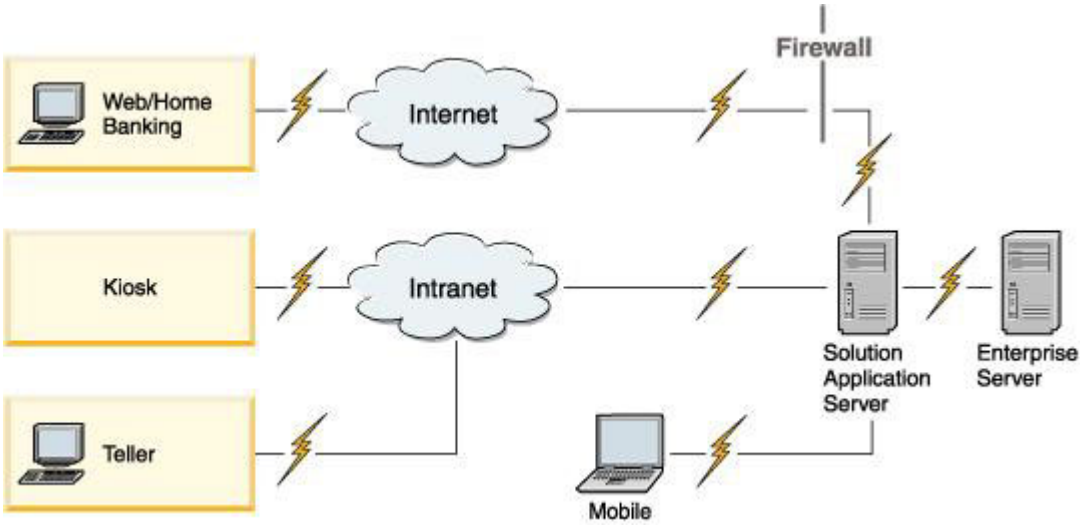
Kiosks and ATMs

The toolkit can be used in kiosks or ATMs that run Internet technologies such as a Web browser and Java. In this environment, the client usually is a Java application (or applet). In addition to the presentation logic, the client application manages the financial devices normally present in a kiosk (such as MSR/E, chip card reader, receipt printer, passbook printer, bar code readers, and touch screen displays) using the financial device services that the toolkit provides. The kiosk connects to the application server using the HTTP or SSL protocols. In some cases, kiosks are located in branches, which handle them as branch workstations. Kiosks can also be connected directly to the server through public or private lines.

Mobile terminal and PDA

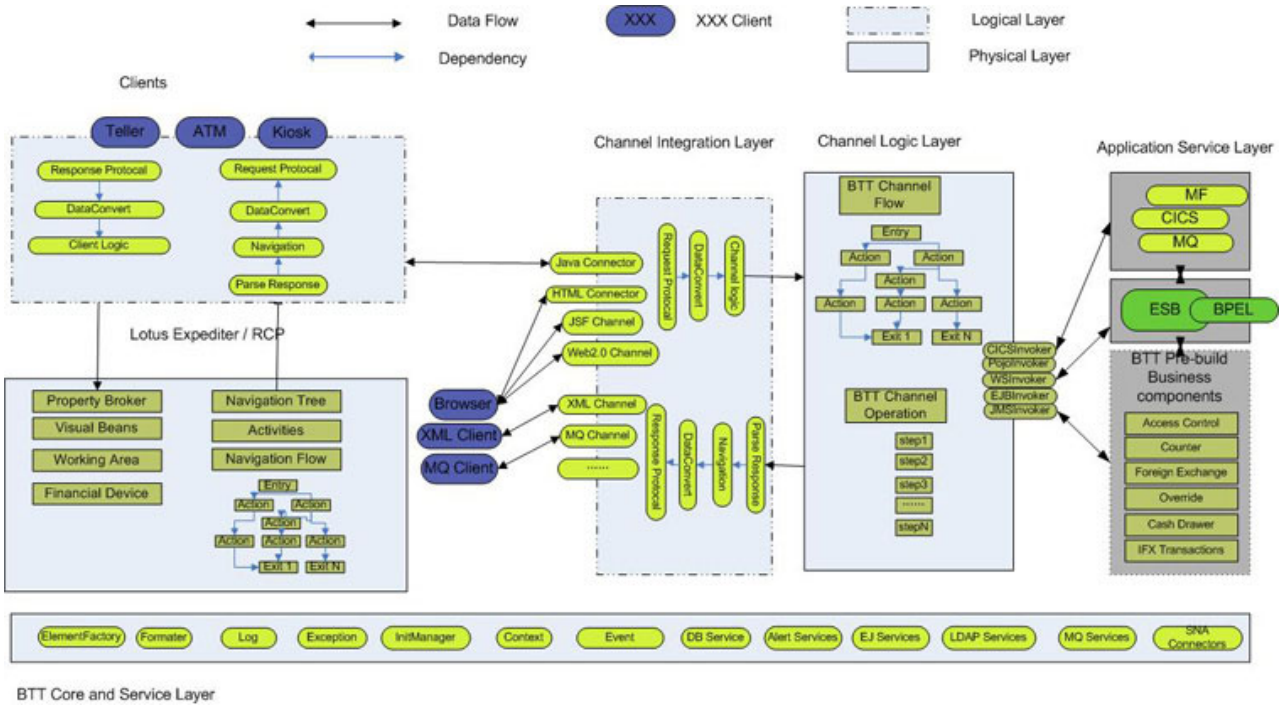
Users equipped with laptops running a Web browser can connect to corporate toolkit servers using the SSL protocol. In this scenario, the toolkit server is usually located at the central site and is protected by a firewall. It is also feasible to have mobile users connected to the branches to which they belong.

The following diagram illustrates these business distribution channels:



Multichannel architecture

The following diagram shows the architecture of the Bank Transformation Toolkit:



Client

A client in the three-tier architecture contains little logic. The logic it does have is usually presentation logic or logic required locally to do such things as accessing financial devices or validating entered data. The code to execute the client logic is downloaded on an on-demand basis, and therefore does not reside on the client, but on a Web server. The Bank Transformation Toolkit supports any kind of physical client device that uses the following technologies:

- Java Client
- HTML Client
- JSF client
- Web2.0 Client

The toolkit provides implementations for current client technologies but these concrete implementations anticipate that significant differences might be found when realizing solutions. The toolkit is not limited to these technologies because its design is generic and can be extended to support other technologies.

Java client environment

The Java client is an application that runs on client desktop platform. The Bank Transformation Toolkit recommended solution is the Rich Client Platform (RCP) which is built on eclipse technology or IBM Lotus® Expeditor. The BTT Rich Client Infrastructure is used for banking customer to rapidly build banking desktop systems. BTT offers an end to end solution to develop Rich Client based teller applications, including transaction development, transaction UI development, transaction panel deployment, application layout management, and so on.

In the following example, the application presentation layer runs dependently on the client side, and the application logic layer run on the WebSphere Application

Server. This example shows how the presentation layer works and uses BTT Single Action EJB (SAE) or BTT Operation to perform the business logic:

1. The user requests a transfer request and provides the required input data:
   a. The user starts up RCP based teller sample.
   b. The user clicks the navigation item named **transfer** in navigation view or input corresponding launch code in quick launch bar to start this transaction.
   c. The user chooses FROM account and TO account numbers.
   d. The user inputs the transfer number and presses **Submit**.
   e. When the user clicks **Submit**, the client creates the transfer client operation and creates a context for it. The client then chains it to an upper level context.
   f. The client operation collects the server required data fields and uses formatter to format context into a String.
   g. The client operation uses CS Client Service to send this formatted String into the Server side.
2. The application presentation layer sends the customer search request to the application logic layer.
   a. In the server, the servlet acts as the request handler that receives the transaction operation.
   b. The request handler un-formats the request String into request operation context, and chains it to the session context.
   c. The request handler calls BTT Server Operation to execute server side business logic based on the context request.
   d. After business logic is processed, the server context is formatted into String as response, and the response is sent to the client side.
3. The client side receives the server response:
   a. The client operation receives the server response.
   b. The client fires an operation replied event to notify other components that this C/S communication has successfully finished.
   c. UI components receive the operation replied event and refresh them. The reply code displays in the transfer transaction panel.

HTML client environment

An HTML client is generally used for a home banking application built to use the Bank Transformation Toolkit. An HTML client can also be used in any other kind of application, such as a bank teller application or a CRMS. The client machine requires only a Web browser to run the application.

When the user visits the start page of the application and logs in, the browser displays a menu or HTML desktop with a list of available processes. A detailed sequence of the events in runtime is as follows:

1. The user requests a customer search and provides the required input data:
   - The user clicks a customer search link in the HTML desktop. This user action sends a request to the server servlet. The request parameters include the name of the FlowProcessor that is required.
   - On the server side, the requested processor is created and initialized. The FlowProcessor, which is in its initial state because it is the first time it has been used, moves to the next state, to which a JSP page is assigned. The JSP page is executed, and it generates the HTML page for the reply. The reply includes some control data, such as the current processor identifier.

- On the client side, the HTML page that is the reply displays a form with input fields for the customer search criteria.
- The user enters the input data and clicks a Submit button. The form data will be sent as an HTTP post request to the toolkit server servlet. The request data contains the FlowProcessor name and process ID as hidden fields, along with the other input data.

2. The Customer Search business operation is executed on the application server.
   - On the server side, the requested processor is restored, and the request data is validated and unformatted on top of the processor context, and the event is passed to the current state of the processor. These actions advance the process flow. A result can be an input event for the current state, or a trigger for the processor to move to its next state. As the entry action for this state, the Customer Search server operation is executed.
   - As part of the execution of the server operation, a customer search transaction request is sent to the host. When the host returns a reply, the message is unformatted into the server operation context.
   - Another step of the server operation updates the Electronic Journal with data from the transaction. The execution of the server operation finishes.
3. The client view displays a list of customers matching the search criteria.
   - The HTML FlowProcessor moves to its next state, which is a JSP page state. The JSP page is processed, which generates an HTML page with the reply. The reply is the list of customers who match the search criteria, which has been updated in the context by the Customer Search operation.
   - The client displays the HTML page that contains the Customer Search results.
4. The user selects a customer and clicks a Submit button, which performs a Customer Details operation.
   - The user action on the GUI sends a new HTTP request. The request data contains the FlowProcessor name, the process ID, and the selected customer.
   - The process flow advances, and the state executes the Customer Details server operation.
5. The client displays the details in a different panel, from which additional actions can be executed.
   - The HTML FlowProcessor moves to its next state, which results in the execution of another JSP page, and generates an HTML reply page. The reply contains customer information obtained from the processor context, which was updated with data from the host reply. The process ends when the user accepts the information and does not perform another operation. The FlowProcessor in the server then enters a final state, which presents the home page to the user.

JSF client environment

The application presentation layer and application logic layer run on WebSphere Process Server so that the example can show how the presentation layer works and uses Bank Transformation Toolkit (BTT) Single Action EJB (SAE) or BTT Operation to perform the business logic.

1. The user requests a customer search and provides the required input data:
   a. The user clicks a customer search link in the menu. The user action sends a request to the JSF framework.
   b. While resolving the value binding expressions of the JSP pages, the request will be forwarded to BTT JSF extension.

c. After the BTT JSF extension resolves the value binding expressions, the request will be returned to the JSF framework. And then the request will be directed to the JSP page.
d. In the client side, the browser displays the HTML page.
e. The user enters the input data and clicks **Submit**. BTT JSF extension resolves the value binding expressions and acquires the data inputted by the user, and then put the data into the BTT context associated with the managed bean.
f. BTT JSF extension invokes the corresponding execution method in the base bean according to the event fired from the JSP page.
g. The base bean delivered by BTT JSF extension constructs a BTT Invoker instance according to the invoker Id, which is passed as a parameter of the `execute()` method, and then the base bean invokes the BTT Invoker to process the business logic.

2. The application logic layer executes the business process:
   a. BTT Invoker invokes the BTT business components, for example, BTT SAE, BTT Operation and so on.
   b. Upon the completion of the business logic, the BTT Invoker returns the processed result that includes the information needed by the presentation tier.
3. The client view displays a list of customer matching the search criteria:
   - The base bean parses the result from BTT Invoker and picks up the BTT context and the outcome and so on, and then propagates them to the backing bean which originally initiates the request.
   - JSF framework directs the request to the JSP page according to the outcome. When displaying the JSP page, the presentation data will be obtained from the BTT context through the BTT JSF extension.
   - The client displays the HTML page.

Web 2.0 client environment

A Web 2.0 client is generally used for a home banking application built to use the Bank Transformation Toolkit. The client machine requires only a Web Browser to run the application. A Web 2.0 Channel is used on the server to process the client's request. When the user visits the start page of the application and logs in, the browser displays a menu with a list of available operations. A detailed sequence of the events in runtime is as follows:

1. The user performs a Customer Search.
   - The user selects the Customer Search operation by clicking a link on the client desktop. This user action sends an XML request to the server servlet. The request XML includes the name of the operation that will be executed.
   - On the server side, the Web 2.0 request handler will parse the XML request and call the operation to execute the business logic.
2. The Customer Search business operation is executed on the application server.
   - On the server side, the request handler will generate the channel context, parse the request XML and call the operation.
   - As part of the execution of the server operation, a customer search transaction request is sent to the host. When the host returns a reply, the message is unformatted into the server operation context.
   - The execution of the server operation finishes.
3. The client view displays a list of customers matching the search criteria.
   - The result will be formatted to an XML and will be sent to the client.

- The client parses the XML that contains the Customer Search results and updates the client view.

4. The user selects a customer and clicks **Submit**, which performs a Customer Details operation.
   - The user action on the GUI sends a new XML request. The request data contains the operation and the selected customer.
   - The Web 2.0 request handler executes the Customer Details server operation.
5. The client displays the details in a panel.
   - The XML reply contains customer information that is formatted from the operation context, which was updated with data from the host reply.
   - The client parses the XML that contains the Customer Details results and updates the client view.

Channel application server

The channel application server includes the following layers:

- Presentation layer
- Channel aware logic layer
- SOA foundation

Application presentation layer

The application presentation layer works in conjunction with a system application server (such as IBM WebSphere Application Server) to provide a layered multiple channel architecture. The application presentation layer works as a bridge that connects the clients with the application logic layer, which performs business transactions. Java clients and HTML clients use different application presentation components to connect to the application logic layer.

To get connected with the application logic layer, the presentation layer defines the following entities:

- **Java RequestHandler** processes a Java client request for a particular type of requester. The toolkit registers these handlers to determine which specific handler it needs for a specific request. For example, there are different RequestHandlers for requests coming from a Java client in a home banking environment, from a Java client in a branch teller environment, and from a Java client in a call center environment. The RequestHandler is responsible for interacting with the client side operations that controls the dialog navigation for a specific client type and for interacting with invokers that call application logic layer transactions.
- **Java PresentationHandler** processes the reply for a particular type of requester.
- **Struts Extensions** processes requests from HTML clients, calls application logic layer components for business transactions, and renders presentation for HTML clients based on the business transaction results.
- **HTML RequestHandler** is responsible for processing a particular request from an HTML client. The handler may need to be aware of the device type. This is managed by the channel context. The request handler performs the following tasks to integrate with the application:
  - Establishes the session between the client and the server for the specific device
  - Executes a generic application operation for the HTML channel
  - Determines the appropriate presentation handler from the handler registry to render the results back to the client.

- **HTML PresentationHandler** is responsible for processing the reply to the HTML client. The main API provided by this class is `void processReply(ChannelContext, ServerOperation)`. This starts the process of dynamically creating the HTML and rendering it to the client using the servlet JSP engine.

To pass business process requests to the application logic layer, the application presentation layer has the Bean Invoker Factory. The Bean Invoker Factory creates invokers so that the requester can invoke the EJBs that perform the business processes in the application logic layer. The requester can be a request handler from the Java client or an EJB Action from the toolkit Struts Extensions component.

Channel aware logic layer

The Bank Transformation Toolkit channel logic layer provides all the options to execute BTT logic related with different channel applications in the application server from applications running in disparate client environments.

The entry points to the application server are different based on the type of client device and the communication protocol being used by the client application. Each of these entry points relates to a specific request handler, which is able to manage channel-specific considerations. To isolate the way of receiving the requests for a specific channel from the application server logic, the toolkit defines some common interfaces to be used by any of the request handler implementations. These definitions are known as the multichannel architecture, and all the connectors listed in client/server connectivity implement the multichannel support.

For specific channel application, such as Internet Banking or Teller system, different channel application has different channel aware logic. For example, the Account Query transaction in Internet Banking can display the final result to user, which only contains basic account information, while the Teller System will show full information, although the two transactions in backend system are the same.

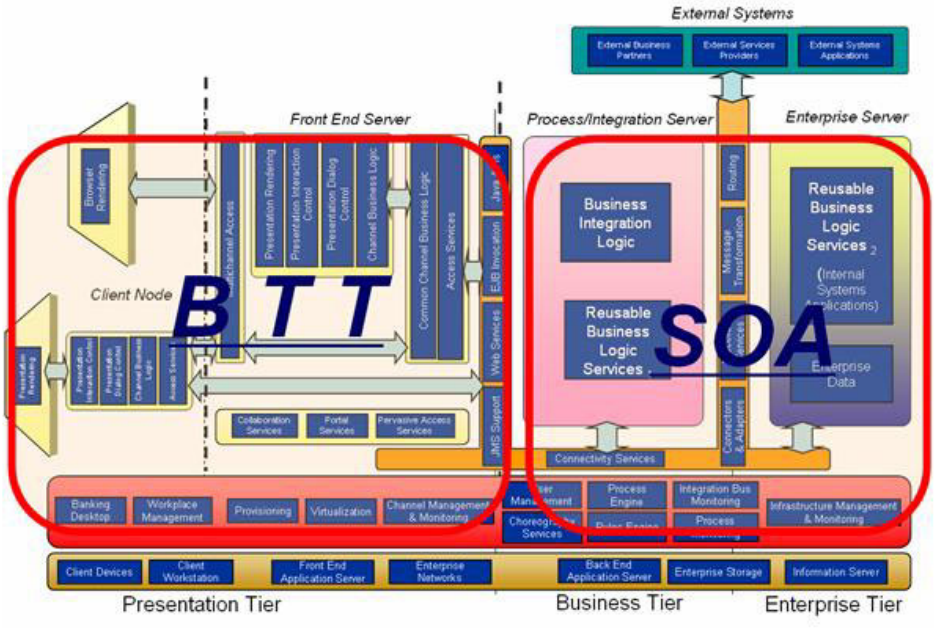
BTT provides channel aware logic layer and associated components, such as channel-aware Operation/Operation Step Definition, channel-aware Processor. You can define Operation/Operation Step and Processor components in both the channel-aware logic layer and the business layer. Note that if you define Operation/Operation Step and Processor components in the channel-aware logic layer, the Operation/Operation Step and Processor will only take actions that are related to the channel logic, but not the reusable channel-independent business logic.

SOA fundamentals

For backend integration, Bank Transformation Toolkit enables your channel applications to support Service Oriented Architecture (SOA). BTT interfaces with WebSphere Process Server (which contains WebSphere ESB) for business process automation and enterprise application integration. WebSphere Message Broker and the WebSphere Business Services Fabric can be added depending on the SOA requirements.

When a complex transaction involves backend Web services, the toolkit supports Web Services JSR 109 standard and it allows Web service invocations from the toolkit's own business layer. On the other hand, the BTT business logic can be treated as a service to be reused by the other application systems. Furthermore, the Web service interfaces of JCA SNA LU0/LU62 connectors are in readiness for the Web service invocation for the legacy connectivity.

The following diagram shows the relationship between BTT and SOA:



The entire banking SOA reference architecture includes the following flow and control concepts:

- Channel Interaction Orchestration
  - Screenflow: A lightweight Web/rich client tier control mechanism (usually a finit-state-machine) that guides the user from screen to screen. States and flows are encoded in XML.
  - Channel Application Microflow: A lightweight Web/rich client tier control mechanism that provides a structured way to organize channel application operations such as screen flows, logging, reusable channel specific logic, invocation of business processes, and invoking back-end services. Tooling is specific to and integrated with the channel application platform. Flows are visually designed and encoded in XML.
- Business Process Automation
  - Macroflow: Long-running process or process involving human tasks to be performed by multiple people. Encoded in BPEL as a linear process or Business State Machines.
- Enterprise Application Integration
  - Service Composition: The creation of a course-grained service from a number of finer-grained services and simple flow logic. Usually created using SCA components.
  - Service Orchestration: Invocation of multiple services in the context of a microflow or macroflow execution. A flow or state machine can be used as the control construct to create a composite service from element services.
  - Routing and Transformation: Routing of a service request to a service provider at runtime according to pre-determined rules and the transformation of the service name, number and type of parameters, and data structures as needed so as to insulate service consumers from service providers.
  - Dynamic Service Selection: Determination of how to resolve a service binding at runtime.

Enterprise Server

The enterprise server, or the back-end server, contains the existing core business logic of the financial institution that is accessed by the toolkit application. A toolkit application does not require changes to such a system or changes to its messaging interface. This is possible because the toolkit includes a rich set of back-end system connector components and message formatters. BTT JCA SNA and Invoker components are provided for SNA (LUA interfaces), JMS, EJB, WebServices, and any other customer extensions.

On the other hand, if you have already built up the SOA based back-end system such as ESB, the toolkit enables your applications to support Service Oriented Architecture (SOA) integration.

Bank Transformation Toolkit interfaces with WebSphere Process Server (which contains WebSphere ESB) for business process automation and enterprise application integration. WebSphere Message Broker and WebSphere Business Services Fabric can be added depending on the SOA requirements. But typically, they do not interface with BTT directly.

Unified invocation architecture

Bank Transformation Toolkit provides a unified invocation architecture.

You can define different types of invoker in the definition XML file. BTT provides unified APIs to get invocation instance from the Invoker Factory and to execute synchronous or asynchronous invocation. Because this framework separates the application code of the invocation and the invocation target definition and parameter, it can provide great flexibility. If you want to change the invocation parameter, or change the invocation type, for example, from EJB to Web Service, the application code does not need to be changed. You only needs to change the invoker XML definition.

BTT invoker framework also supports multiple XML files of invoker definition. Invoker Factory is an instance factory. It can have a lot of instances and copies in memory. Each Invoker Factory represents one XML definition. You can query Invoker Instance from the factory by the ID defined in the XML files.

This unified invocation architecture can invoke the following types of target:

- POJO
- EJB
- WSProxy
- Web Service DII
- JMS

Components architecture

The Bank Transformation Toolkit contains the following components:

- Core components
- Presentation components
- Service components
- Business components
- Development tools
- Runtime tools

- Support tool

Core components

The Core components are the main entities of the IBM WebSphere Multichannel Bank Transformation Toolkit.

The core components includes the following:

- Operation
- Flow
- Formatter
- Data element
- Initialization manager
- Element factory
- Invoker
- Exception
- Events
- Externalizer
- Trace

Presentation components

The Bank Transformation Toolkit provides components that facilitate the construction of the client presentation logic. The toolkit includes the Rich Client infrastructure, a set of SWT Visual beans, as well as the support for JavaServer Pages (JSPs).

The following are the provided presentation components:

- Rich Client infrastructure, which offers pre-build components to save your effort on developing teller systems.
- SWT Visual beans, which is based on Rational® Application Developer, Visual editor and DSE Visual Beans.
- JSP support, which includes custom JSP tags and utility beans to enable applications to retrieve information from the operation context hierarchy, get resources, and handle errors. If your application requires additional behavior, you can build new tags using the JspContextServices interface.

Service components

The Bank Transformation Toolkit provides a set of service objects that enable an application to complete an operation. These services include host communications, journaling, store-and-forward for off-line operations, financial devices for input and output operations, and more.

The service components include the following:

- Communication services:
  - JCA LU0
  - JCA LU62
  - MQ connector
- Database services:
  - Database Table Mapping
  - Electronic Journal

  - Store
- LDAP Access service
- Generic Pool
- Financial Device services:
  - Check Reader Device service
  - JXFS service
  - LANDP® MSR/E Device service
  - WOSA Device service

Business components

The business components include the following:

- Foreign Exchange
- Cash Drawer
- Counter

Development tools

The Bank Transformation Toolkit provides a number of tools that support the development of applications. The Bank Transformation Toolkit provides a number of tools that support the development of applications. All the tools are plug-ins of Rational Application Developer (RAD) and WebSphere Integration Developer (WID).

The development tools include the following:

- Formatter Simulator
- Migration Tool
- SWT Visual Beans Editor
- Transaction Editor
- Validation Tools

Runtime tools

Runtime tools are used to fetch or record information of the BTT runtime.

The Runtime tools include the following:

- Runtime Monitor tool
- Trace Facility

Support tool

The Support tool includes the following:

- APAR Tool

Development

The Bank Transformation Toolkit is developed using Rational Application Developer or WebSphere Integration Developer. The Bank Transformation Toolkit consists of a set of tools that support end-to-end development and deployment of e-business applications. It facilitates development tasks such as rapid application development, creating industrial-strength Java programs, and maintaining multiple editions of programs.

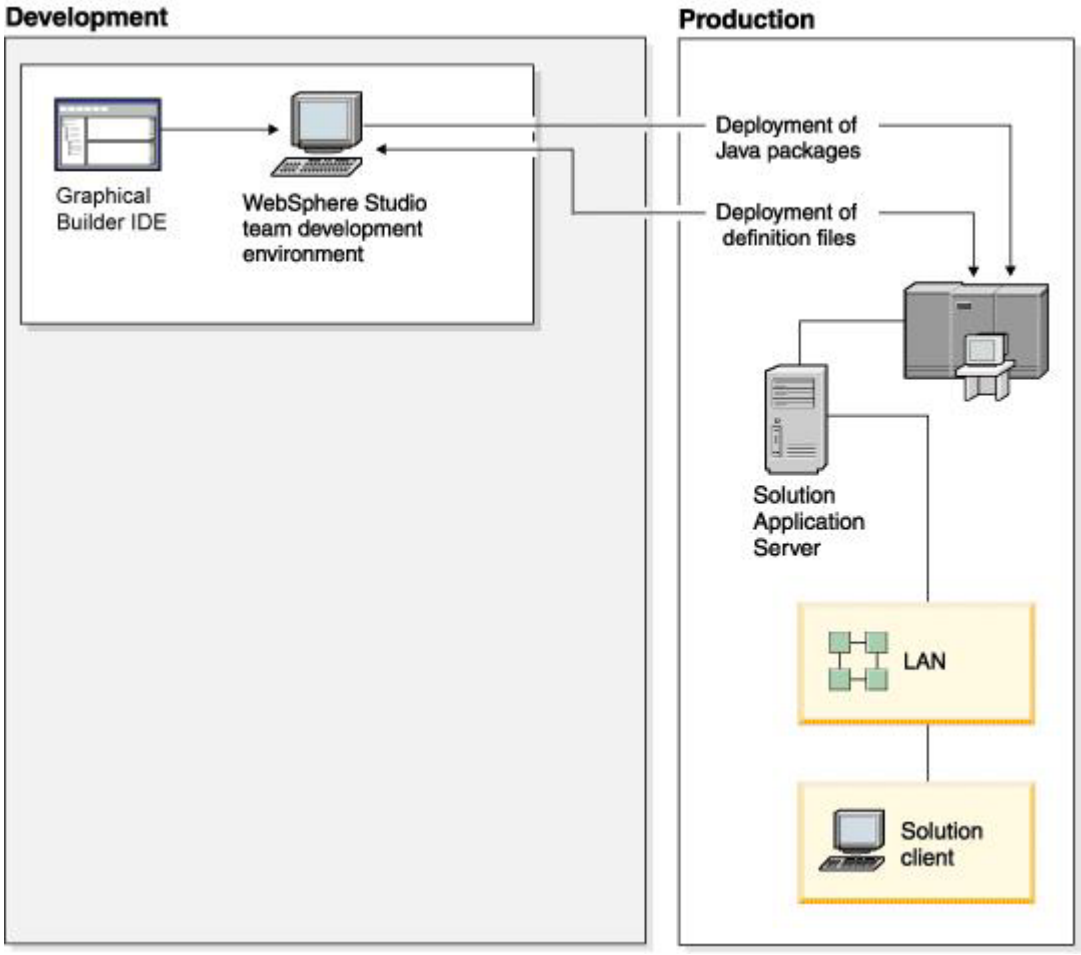
The toolkit provides a set of components built as Java classes and JavaBeans™. The method signatures and class definition of a bean follow a pattern that permits visual development environments to determine the bean's properties and behavior.

The following development tools can be used to build a solution using the Bank Transformation Toolkit:

- Formatter Simulator
- SWT Visual Beans Editor
- Transaction Editor
- Validation Tools

Development Model

The Bank Transformation Toolkit proposes a repository-based development model where all the relevant information about financial transactions (data, formats, contexts, services, processors, and views) is externalized to a set of definition and configuration files and separated from the Java code. The development model allows developers to add new processes or transactions in a toolkit-based application with minimum coding required, by adding some definitions into the definition repository. The following diagram depicts the role that the definition repository plays in the development and deployment of a toolkit-based solution:



This separation allows parallel resources to be focused in each of the areas, but it requires a common understanding and definition of the model to get a final consistent solution.

Development process phases

During the overall development process four main phases are identified:

1. Analysis and design.
2. Coding reusable entities.
3. Code the runtime application and feed the repository.
4. Generate the runtime resources and test.

The phases are iterative and contain tasks that may be refined during the project life cycle. For information on the tasks within Bank Transformation Toolkit development and where you can find more information on these tasks, see Creating an application with Bank Transformation Toolkit.

Physical deployment

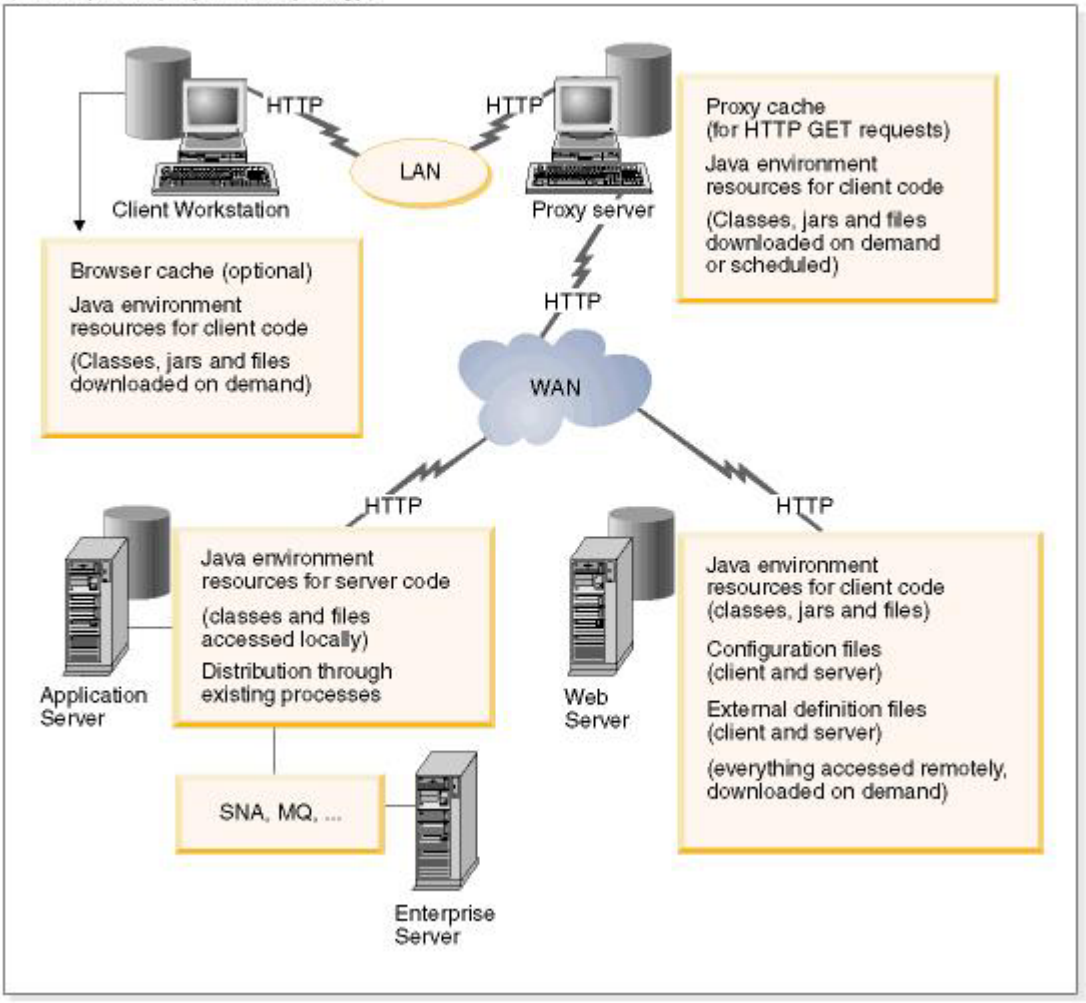
A toolkit-based application should use the standard mechanisms of the Internet or network computing technology for the distribution of objects and should exploit the cache mechanisms to get the best response times.

The physical location of the toolkit components depends on the particular project environment and requirements. The classes and required resources for the toolkit components, such as configuration files, definition files, and icons, may reside either on the local workstation where the application is executed or on a remote server being accessed through HTTP. Its resources are drawn from two main sources:

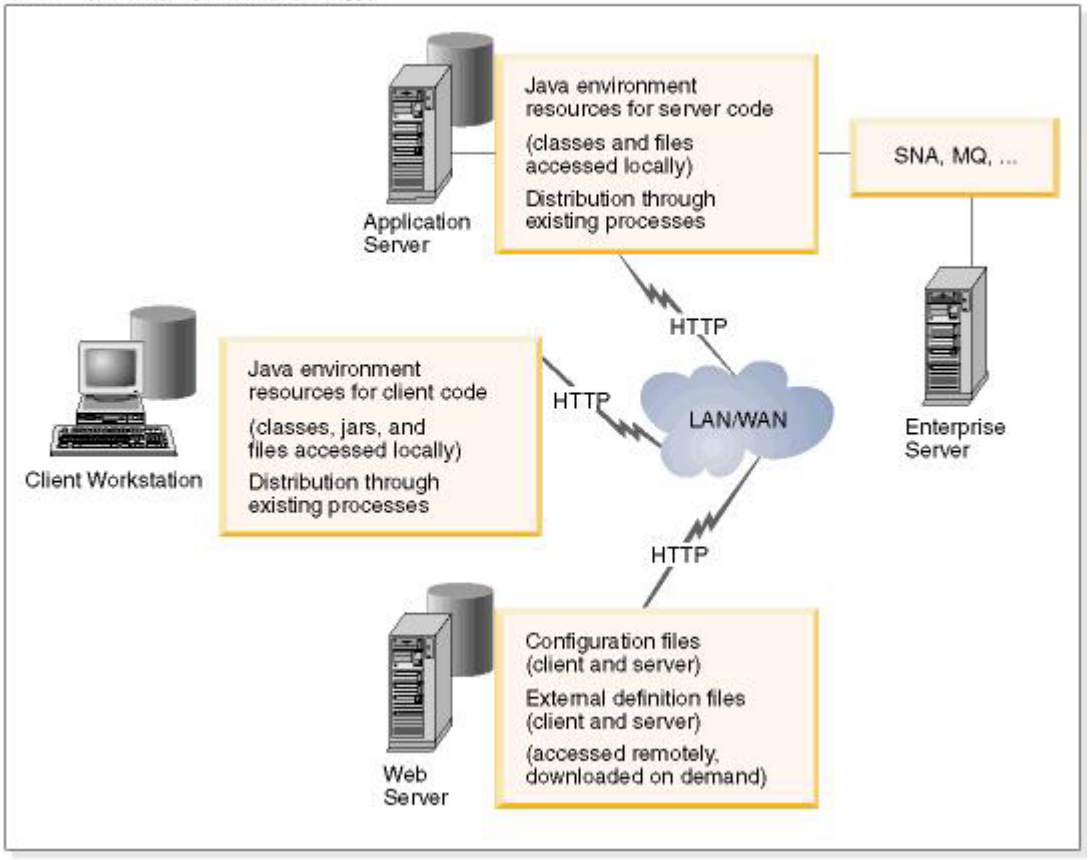
- The classes with their corresponding resources, obtained from the Java resources environment; that is, loaded by the existing class loader following the active classpath. This will be done regardless of whether the code is executed as an application (resources will be located locally) or as an applet running inside a browser (resources will be located either locally or remotely).
- The required configuration and external definition files, as specified in the user settings of the toolkit environment. These allow the resources to be located either locally or on the server, regardless of whether the code executes as an application or as an applet running inside a browser.

The following diagrams depict sample deployments, with the application code being executed inside a Web browser in the first diagram, and with the application code being executed as an application in the second diagram. Note that the application server and the Web server are different logical entities and can be located in different physical locations, although in simpler configurations, they would coexist in the same server.

Code running inside a Web browser
(a sample deployment topology)



Code running without a Web browser
(a sample deployment topology)



Cache mechanisms

An Internet or network computing architecture has the required installation base code and resources in a central location. Nothing is installed on the client workstations and the central location distributes the required resources on demand from the Web server through the communications network for execution on the client. This topology requires high-speed communication lines, and is enhanced by the use of cache mechanisms in the Web browser and in the proxy servers. The cache mechanisms allow the reuse of objects previously distributed, thereby reducing the requirements of the physical transport layer.

Cache refresh policies

The use of caching requires a refresh policy that prevents the executing application from using out-of-date versions of objects in the caches. Proxy servers can be scheduled to refresh their caches automatically at a prescribed interval or on demand at a particular time. All refresh policies are based on actions started in the proxy server, either because an object has expired or through scheduled processes for checking the versions of server objects. There are no dynamic updates of objects in the proxy when the objects are updated on the Web server. Therefore, depending on the specific system environment and the detailed analysis of the proxy server features, the issue of consistency between the proxy and the server must be resolved, and its solution built as part of the application process.

JAR files

Part of a physical deployment strategy is to set the policy for packaging the code and the resources for an application, as well as to decide the locations for code and resources and their distribution to the final destination workstations. A solution based on the toolkit may use Java Archive (JAR) files, which provides a physical packaging mechanism for a set of HTTP objects or resources (including classes and files). The JAR files have the following advantages:

- Reduce the number of interactions with the server during the download process of the resources
- Compress the objects, thus improving the performance in the transmission and the memory optimization in the cache of the browser

The following packaging considerations regarding JAR files enter into finding a satisfactory balance between the number of objects to handle and the desired network performance:

- The number of JAR files
- Grouping objects that are used when a specific business function is executed
- Grouping objects on the basis of likelihood or frequency of change
- Size of the JARs

Application components and toolkit components may be packaged in JARs, since JARs can be used to package not only the Java classes but all the configuration and external definition files required by the solution.

Bank Transformation Toolkit best practices

This section introduces the best practices, programmers' tips, and performance tips of Bank Transformation Toolkit:

Best practices

This section contains a set of recommendations and best practices to be used when developing Bank Transformation Toolkit applications.

BTT Context tree

The BTT Context tree is a central piece of the framework. A proper design of the context tree is critical for the maintainability and robustness of the application. The following recommendations are given in relationship with this subject.

Session context size: This section contains recommended best practices on how to minimize the allocation of data in the BTT session context. The BTT session context stores the data and services a given user needs across several requests. It is usually the major consumer of memory resources in the server. The average size of this data determines how many users can be allocated in a given JVM; thus highly impacting the application's scalability. Moreover, if session persistence is used, for example in order to enable transparent failover, the session size becomes more critical because it greatly impacts performance, given that all the session data will be serialized at the end of each request.

The recommendations for minimizing the session context size are as follows:

- **Session size profiling**: Calculate the average session size of the application in order to ensure that the available hardware can allocate the planned number of users.

A profiling done with tools such as LoadRunner or Rational Application Developer (RAD) will help determine the exact size of a given session instance. Whereas, WAS Performance Monitoring Infrastructure (PMI) can directly report the average HttpSession size in a server.

- **Session size capacity planning**: The following calculations should be performed for each WebSphere cluster where BTT is planned to be installed:
  - Determine the memory available to the application (JVM size) per server.
  - Keep failover in mind. In the case where one server in the cluster is down, the rest of the servers should still be able to allocate the extra resources. This means that you must account for one server less than the total number of servers available.
  - Divide the total available memory by the planned number of users.

  This will result in a rough estimation of the maximum session size. If, after profiling, the average session size is close to this maximum, the application should be modified in order to reduce its session size.
- **Reducing session size**: The following tips can be applied when designing or reviewing the session context.
  - Ensure that all data defined in the session context is really necessary at that level:
    - The data is user-specific, and therefore cannot be placed on a higher-level, shared context.
    - The data is required along several user interactions, and therefore cannot be placed at a lower-level, operation context.

    If any of these two requirements is not present, then the data can be placed at a different level, reducing the total size of the session context.
  - If a given data entity is required at the session level, analyze the usage pattern of this entity:
    - If the data is seldom used and the performance impact of retrieving it from the source is not too big, consider defining it at a lower-level context.
    - If the data is used throughout several steps of an interaction involving multiple pages, implement a mechanism for creating the data structure at the beginning of the interaction and destroying it at the end. This can be done by defining data at the flow context level, or by adding data to the session context (for example through a dynamic keyed collection) and removing it at the end of the interaction.

If session persistence is enabled, the following recommendations should also be considered in order to reduce the performance impact of the persistence process:

- Persistence performance impact is mostly caused by the serialization process. It is therefore very beneficial to provide a custom serialization implementation for the data being serialized. With the BTT framework, this means usually only the DataField, KeyedCollection and IndexedCollection will need to be extended, since all context data is stored in this kind of structures.
- Instead of serializing all session context data after every request, consider developing an extension that marks which objects have been updated, then design your persistence database so that each different session object is stored in a separate field. This approach can have great performance benefits, but is difficult to implement and may require active participation of the application code, that is to notify whenever an object has been modified so it can be appropriately marked for serialization.

Shared contexts: Shared contexts are those contexts above the session contexts. These contexts are potentially accessed from multiple threads corresponding to different user requests at the same time, so some considerations have to be taken into account:

- **Read-only data**: only place the read-only data in shared contexts. There are no concurrency problems when accessing read-only data, so this is perfectly safe.
- **Read-write data**: in the case where data needs to be updated during the execution of the application, consider the following two problems:
  - **Concurrency**: because multiple threads can access the same field in a shared context, there are concurrency problems if the data are not accessed with proper synchronization safeguards.
  - **Server clustering**: when more than one server clone is used, there is a copy of the shared context tree for each JVM where the application runs. If a user's request requires that a data field in a shared context to be updated, its new value will only be visible to the users whose session is located in the same server where the original request was executed. This is probably not what is expected, as shared values are designed to be transparently accessed by multiple users independent of the clone they are running on. Consider for example the case of a branch-level unique receipt counter that gets incremented every time a receipt gets printed. That could be in principle located in a branch context, but then two users of the same branch whose session runs in different clones will manipulate totally independent receipt counter values.

Both the concurrency and clustering problem can be solved by using a safe persistence-based mechanism to manipulate all read-write shared data. A simple, ad-hoc approach is to use a custom database table to access these values. There is also a more generic way supported by BTT since version 5.2: the shared data can be set up so that they are automatically persisted by the framework. This is known as the CHA (Context Hierarchy Area), and since version 6.1, there is a high level of flexibility in the way that this can be configured, either through entity bean persistence, shared memory or any other user-provided mechanism.

Application complexity management

This section contains recommendations aimed at reducing the complexity of the application, thus increasing its manageability and maintainability.

Naming conventions: Development is basically a process of creating a big amount of code artefacts such as Java class, JSP pages, XML files, and in the case of BTT, XML tags with an ID attribute. For maintainability reasons, it is very important that clear naming conventions are documented and followed throughout the development process. Which conventions are used is not so relevant as long as they are consistent and based on common sense.

In order to ensure that the conventions are followed, a process to verify the naming convention rules can be added to the code quality review toolbox used in the project.

BTT application grouping: The recommendation is actually not BTT specific, and it can be applied to any J2EE application: avoid packaging and deploying a system as a single WAR/EAR package. Try to break the application into smaller functional groups based on dependencies, function set, development and maintenance groups, etc. Apply the standard J2EE and WebSphere recommendations on this subject, and consult an expert if required.

Breaking a big project into smaller projects renders it more maintainable, easier to test and administer on the runtime environment, and isolates any failures in the failing subprojects.

XML management: The size and manageability of BTT XML files has to be controlled: if all the XML code resides in a few large XML file, there is no easy way to allow several developers to edit the same file in parallel, since committing the changed code to the repository will require a complex merge operation of the changes made by each developer.

That is why it is recommended to use BTT XML mixed modularity. The operation and process instances can be placed each in a separate XML file, and the global, shared definitions (such as the higher-level contexts and all their data and services) are placed in the root XML files.

Using an XML validation and review tool is also recommended: this can range from simply using the provided BTT DTD/Schema files in the XML editor, to applying a custom-made quality review tool that verifies naming conventions, searches for dead unreferenced code, and checks any other project-specific rules.

Even with these rules in place, there is still a lot of XML code to be managed. Some other recommendations are the following:

- If even with a mixed modularity approach, the size of some XML files is too big, consider splitting these files into smaller chunks. The files can be easily merged at server startup time, just before instructing BTT to process the XML file.
- Apply the recommendation given above: separating a project into a smaller WAR/EAR deployment units. For example a big project with 1000 XML files is split into four independent applications, each application will have an average of 250 files, which is a more manageable figure.

Tracing

The following recommendations can be applied to the BTT framework and application traces. Most of the recommendations are not BTT specific, but can be applied to any tracing implementation:

- Consider using Aspect Oriented Programming to de-clutter the application and infrastructure code. In some cases, around half of the code might be trace code that complicates its understanding and therefore its maintenance. Naming conventions are important when using AOP since they help define clear AOP rules.
- Avoid by all means tracing to the system console through System.out or System.err. This cannot be switched off easily and therefore it will still execute in the production environment.
- Add a check before each trace to verify if the traces are enabled for the provided level and component ID, only trace if the check returns true. Strictly speaking, this is required only if the message string is generated by executing some concatenation or rendering code, in order to avoid the useless execution of that code. If AOP is not used, the check might be skipped when tracing simple predefined messages, in order to reduce code clutter.
- Use the appropriate trace levels, and then review the high-severity messages. A step of the quality review process can consist of reading the traces and ensuring that no trace above the warning level is ever generated in otherwise correctly working code. If a high severity trace is detected, it should be fixed through one of the following actions:

  - Either the code is really failing and needs to be fixed until no trace is generated, or
  - The trace level for that component is incorrect and needs to be lowered down.
- In the production environment, disable all the trace levels except the highest severity ones. Tracing can degrade performance: if required, use the pre-production environment, which should have more traces to locate problems.
- Do not mistake technical-levle tracing, which should be used exclusively by developers to pinpoint potential problems, with business level-logging. If you need to keep an audit trail of the execution of the business function, use a separate system, and implement it through a BTT service.

Separation of infrastructure and application code

In the past, J2EE did not provide all the necessary components for an end-to-end application architecture, so a framework was much needed. Frameworks such as Struts, Spring or BTT had to be used to provide the extra functionality, mainly:

- Rich web component-based rendering
- View componentization and reuse
- Presentation flow management
- Data definition, validation, conversion, and mapping
- Business flows / Integration flows
- Back-end connectors

The present set of J2EE standards in combination with world-class WebSphere products now provides very standard and robust implementations for each of these functionalities:

- Rich web component-based rendering: JSF
- View componentization and reuse: JSR 168 Portlets / IBM Portal
- Presentation flow management: JSF
- Data definition, validation, conversion, and mapping: JSF
- Business flows / integration flows: BPEL / IBM WebSphere Process Server
- Back-end connectors: JCA

However, there is still much need for a framework: the focus of its importance is no longer in the functionality that its components provide, but in the order it brings to the complexity of J2EE. With only J2EE and standard development tools, an application developer needs to continuously make architectural decisions, because there are many ways to develop a given functionality, and many possible patterns to apply. This approach would pose a risk in the robustness and maintainability of an application. Even if all developers of a project were highly skilled J2EE architects, there should be a consensus on which patterns to apply, and these should be respected all over the application, otherwise, the application would be hard to maintain.

BTT is a framework that solves this problem by providing a well-defined architecture for an application and a set of patterns that can be applied in a repeatable way. However, BTT still needs to be adapted and extended to fit the target environment. The run-time and development architecture, BTT extensions, customized tools, and development patterns need to be carefully designed and implemented by skilled developers, as it will be the infrastructure used in the rest of the project. This infrastructure can then be reused in other projects and gradually improved according to changing demands.

This infrastructure is developed in two phases:

- An initial phase where the basic patterns are laid out according to the architecture phase, providing the necessary code and tool extensions for it. The functionality provided should enable the development of a relatively simple but real part of an application, which can be developed alongside the infrastructure extensions; both infrastructure and sub-applications are used to test each other. At the end of this phase, normal application development can begin.
- During the rest of the project, the infrastructure is enriched, driven by the demand of developer. The infrastructure grows in parallel with the rest of the application.

This approach divides the project team in two groups:

1. Standard developers, who make use of the infrastructure ″as-is″, or make simple extensions to it if required. Most of the developers in a project fall into this group.
2. A reduced group of architect-developers, who are continuously in contact with the standard developers, and expand the infrastructure driven by the project demands. The expansion points are decided under the criteria of productivity and robustness improvements.

When a standard developer needs a feature not yet available in the infrastructure, he or she should request it from the architect developers. If for any reason, he or she develops the extension himself or herself, he or she should submit the code to the architect developers for review and proper incorporation into the infrastructure codebase.

The best way to enforce this practice is by clearly defining a narrow set of code artefacts that the standard developers are allowed to create. Examples of such artefacts are BTT XML files, JSP files, and maybe a small set of Java classes extending from well-defined superclasses and with a strict code size limit. Declarative code such as XML is easy to constrain, which ensures that the developer is following a given set of rules. On the other hand, imperative code such as Java is dangerously versatile. Therefore, the advantages of declarative constrained code are the following:

- Ensure that the developer is following the rules set by the architecture team, since each declaration must comply with a set of constraints.
- It is easier to maintain, as all its artefacts fit into a given predefined pattern. BTT has many examples of this: Formats, Operations, Contexts, and so on.
- It is easier to manipulate through tooling, as parsing and representing its structure in memory is easier than doing the same task with a full-fledged imperative programming language.
- Migration is simpler, since parsing and transforming the application code is easier as compared with an imperative programming language.

This is the model followed by many banks using BTT, which has been extended to match the customer's particular requirements. Typical developer group sizes are 20 to 50 standard developers against 3 to 8 architect developers. The number of people in the latter group usually diminishes as the project matures.

Programmers' tips

Following are the tips for programmers:

- Channel:

There is a default rule in BTT Channels to define the data formatter and response formatter.

If you do not specify them in the request, BTT uses csRequestFormat and csReplyFormat configured in operation as data format and response format.

The operation configuration file is as follows:

```
<QueryStockOp.xml>
<operation id="QueryStockOp" context="stockCtx" implClass="com.ibm.btt.poc.opstep.QueryStockOp"
<refFormat name="csRequestFormat" refId="stockFmt" />
<refFormat name="csReplyFormat" refId="stockFmt" />
</operation>

<fmtDef id="stockFmt">
<fXML dataName="stockCtxData">
<fString dataName="code" />
<fString dataName="price" />
</fXML>
</fmtDef>
</QueryStockOp.xml>
```

- Migration:

  During migration, if you do not set the migration rules for the migration tool, the migration tool will set the BTT Version 4.3 to BTT Version 6.1 migration rules as the default rules.

- Trace:

  BTT trace must be initialized at first before the applications use it. Otherwise, the default trace configuration is set and the trace configuration in btt.xml will not take effect.

  If BTT is not initialized, the default trace target is WAS by using `BTTLogFactoryToWASImp` as the implementation class of BTTLogFactory.

- CHA mode:

  When passing local mode CHA context across JVM, only the current context is serialized or deserialized. The parent and children of the current context are not serialized or deserialized. If you want to obtain the data of the parent context, you need to transfer the current context and its parent context separately.

- Invoker:

  It is recommended to generate the definition information file of the Web Service before runtime when using the Web Service DII invoker. This will improve performance and help problem determination.

- Element Factory

  ElementFactory is an implementation of IoC (Inversion of Control) Container. You can use it to apply Dependency Injection pattern in your application.

  Following are the best practices in applying Dependency Injection:

  1. Follow three phases in developing your services or components:
     a. Startup:

        To startup your component, you must set up the configurations and dependencies of your component. Using setter injection helps you to make your configuration file as simple as possible.

        You can implement `initialize()` method in your component. In this method, you can check the configurations and dependencies and you can also allocate required resources in this method.

        You can implement `com.ibm.btt.element.LifeCycle` interface to enable lifecycle support. Or you can define `InitMethod="destroy"` in the XML definition to enable lifecycle support for your element. The benefit of implementing the `LifeCycle` interface is that the ElementFactory will call

`init()` and `destroy()` method. As a result, you do not need to add the `InitMethod="initialize" DestroyMethod="destroy"` definition and your services or components will import BTT classes.

b. Handle requests:

After your component starts up, you can call the business logic in your component. Do not name the business logic methods after `get**` and `set**`, because `get**` and `set**` are used in startup phase.

c. Tear down:

Destroy all the allocated resources in the `destroy()` method.

2. Choose stateless style instead of stateful style:

Choose stateless style whenever you can.

3. Choose singleton scope instead of prototype scope:

Singleton is only applicable to stateless style.

Performance tips

The performance tips given in this section are intended to help Bank Transformation Toolkit solutions achieve the best performance results. A solution architect should decide, based on the solution design, which of the following suggestions apply.

- **Object cache**:
  - The caching of formatters and operations is enabled or disabled in the configuration file. However, the application must exploit this feature by returning objects to the cache.
- **Configuration file**:
  - The toolkit expects some configuration settings to be available in the configuration file. If these settings are not available, internal exceptions are thrown and trace entries are generated, thus consuming CPU cycles even though the default values are still used. When migrating existing applications to an environment where a new product release is installed, consider reviewing the provided configuration file and identifying the changes. Also, consider removing any setting not required in your solution from the configuration file.
- **Data access:**
  - Avoid using wildcards when using the `getValueAt` access method. Use complete data element's paths instead.
- **Synchronized code**:
  - The application flow definitively needs to synchronize those critical code lines when they are executed from concurrent threads (such as arranging the context hierarchy). However, big chunks of synchronized code lines may represent a bottleneck in the solution and reduce the overall throughput.
- **Services access and pooling**:
  - Usually, a solution seeks to improve performance when launching business operations after logging on. It is therefore good design to perform as many processes as possible during the initialization of the services, during the session establishment or user logon, so that the actual business processes execute as quickly as possible.
  - To avoid bottlenecks while accessing services that cannot be re-entered from concurrent users/threads, use services pools. The number of services in the pool must be sized according to the expected load rate. Correct sizing will have a definitive impact on the overall solution throughput.

- **Formatter decorators**:
  - When a record formatter definition includes many formatter entries followed by the same kind of decoration (such as a fixed ″#″ as a delimiter), consider extending the formatter class to include the decoration inside the format process. This mechanism will create only one object (usually a StringBuffer) instead of several strings.
- **Exceptions that are part of the normal flow**:
  - Avoid exceptions that are normal during the application execution flow (such as DSEObjectNotFound).
- **Extended classes to be customized**:
  - Classes available in extension packages (such as com.ibm.btt.automaton.ext and com.ibm.btt.base.types.ext) are especially provided to be further extended in a solution. Consider extending these classes both to add your own logic and to remove non-required logic.
- **Client/Server Mechanism**:
  - Consider using a compression decorator in the client/server request and response formatters to minimize the amount of data sent through the communications network.
- **JSPs**.
  - Use JSPTags and do not use JSPBeans.
  - Consider a solution based on an XML-formatted data set being returned to the client and processed by a template processor in the client (XSLT). The corresponding request handler may be extended to build a faster stream based on formatters instead of JSPs. This approach requires less network bandwidth and is faster than building the response on the server. However, it has other implications that need to be considered such as the XSL support in the Web browsers.
- **Deployed JARs**:
  - Choose only the JARs that belong to the components that are used in the solution. Keep JAR files granular and as small as possible.
- **High availability, load balancing, failover, and session persistence**:
  - 24x7 available solutions have a very high performance or monetary cost. Consider using load balancing with session affinity, so that once the user establishes a session with a server image or clone, all the requests will be routed to that clone.
- **Trace**:
  - Use the `BTTLog.doXXX(doDebug/doInfo/doWarn/doError/doFatal)` method as a Boolean condition for tracing in the application flow, to check whether the system will trace the entry based on the external configuration. The application will only create the string if the returned Boolean for the doXXX method is true.
- **JDBC Table access services**:
  - Consider using stored procedures when requiring access to several tables in the application flow. Cross-logic against several tables using many JDBC Table access calls is not recommended.
- **Java Profiler**:
  - Identifying the objects that are created most often and the classes and methods that use more CPU time during the request process is crucial to optimizing the solution performance. Any Java profiler may be used to get

this information, and this is a task that should be done during the whole development cycle, without waiting until the final implementation of the solution.

Supported platforms and technical requirements

This section presents the supported platforms and software required by each of the Bank Transformation Toolkit components. Because the toolkit is built in Java, any additional platform that provides the corresponding Java Virtual Machine is supported by the toolkit architecture.

A new solution with additional platforms may require changes to the toolkit to make it more generic so that the new solution can cope with the current platform as well as the new one. These changes may involve enabling the toolkit interfaces or components to support the new platform, and may be required for both the hardware and software components of the solution. In cases where native interfaces are required, a gap analysis is needed to support the new specific modules not provided by the toolkit. The components that are actually used depend on the specific requirements of each customer.

Components and platforms

In the following table, an X indicates that the service or component can be installed on that particular platform. Note that nothing prevents an application from accessing a service or component installed on another platform.

Table 1. Java client components

Component name		Windows® Server 2003	Windows XP	Linux® Intel®
BTT Core		X	X	X
BTT Visual Beans		X	X	X
BTT Rich Client		X	X	X
Financial devices	WOSA/XFS	X	X	
	J/XFS	X	X	X

Table 2. Application server components

Component name		Windows Server 2003	Windows XP	AIX®	Solaris	Linux Intel
BTT Core		X	X	X	X	X
CHA		X	X	X	X	X
BTT Channels		X	X	X	X	X
BTT Business Components		X	X	X	X	X
BTT Database Services		X	X	X	X	X
BTT Invoker		X	X	X	X	X
BTT Services	LDAP Service	X	X	X	X	X
	MQ Service	X	X	X	X	X
Communications	JCA LU0 or LU62	X	X	X		X
Database services	Database Table Mapping	X	X	X	X	X
	Electronic Journal	X	X	X	X	X

Table 3. Tools

Component name	Windows Server 2003	Windows XP	AIX	Solaris	Linux Intel
BTT Runtime Monitor	X	X			X
BTT SWT Visual Beans Editor	X	X			X
BTT Transaction Based Tool	X	X			X
BTT Validation Tool	X	X			X
Formatter Simulator	X	X			X
BTT Migration Tool	X	X			X

Components and technical requirements

The following table shows the additional technical prerequisites of the Bank Transformation Toolkit components. For version information, see the Installation Guide.

Table 4. Additional technical prerequisites

Component name		Technical requirements
Financial device services	WOSA/XFS	WOSA/XFS manager and device-specific SPM
	J/XFS	J/XFS manager and specific device service
LDAP Service		IBM Tivoli® Directory Server
MQ Service		IBM WebSphere MQ
Tools		Rational Application Developer 7.0.0.5 or WebSphere Integration Developer 6.1.0.0

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Lab Director
IBM China Software Development Lab
Diamond Building, ZhongGuanCun Software Park, Dongbeiwang West Road No.8, ShangDi, Haidian District, Beijing 100193 P. R. China

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java is a trademark of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.