IBM WebSphere Multichannel Bank Transformation Toolkit
Version 8.0

Solution Architecture

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 19.

This edition applies to Version 8, Release 0, 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.

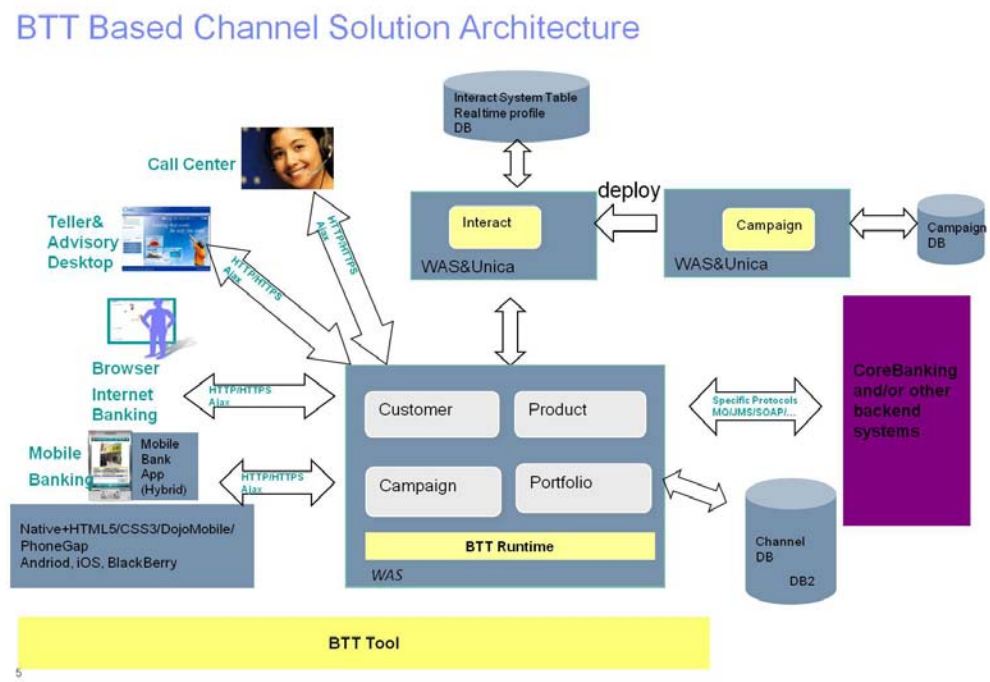
Readers of this document are assumed to be familiar with object-oriented software and related development techniques, and to have a general knowledge of J2EE and related technologies, network computing, and Internet technologies.

Introduction

IBM WebSphere Multichannel Bank Transformation Toolkit is a component-based toolkit for developing enterprise e-business applications. WebSphere Multichannel 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 WebSphere Multichannel 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, WebSphere Multichannel 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 WebSphere Multichannel Bank Transformation Toolkit runtime architecture is based on the J2EE architecture with extensions, and many development tools the toolkit provides are Eclipse plug-ins.

The typical BTT based solution architecture shows in figure 1:



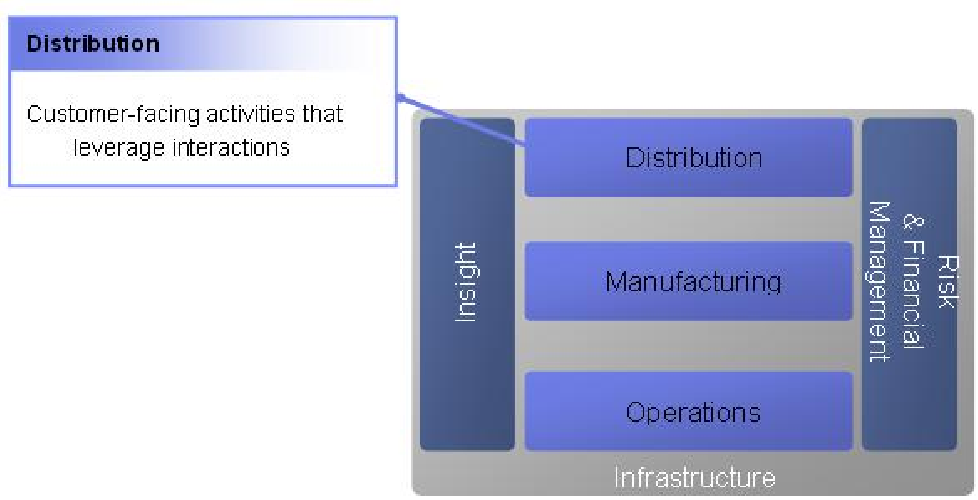
The Value of Bank Transformation Toolkit to Business

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

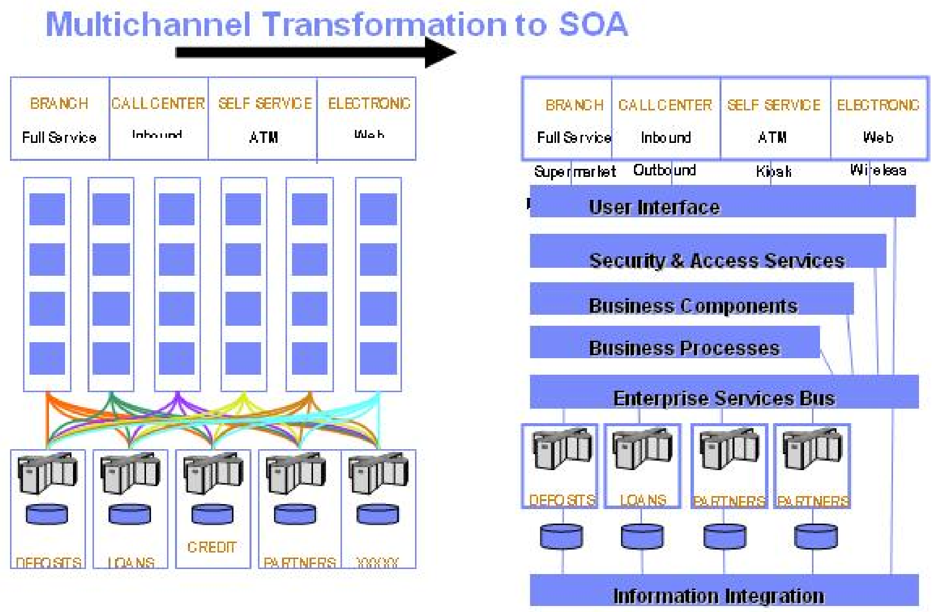
Bank channel applications enable nearly all the customer interactions.

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

Multichannel transformation

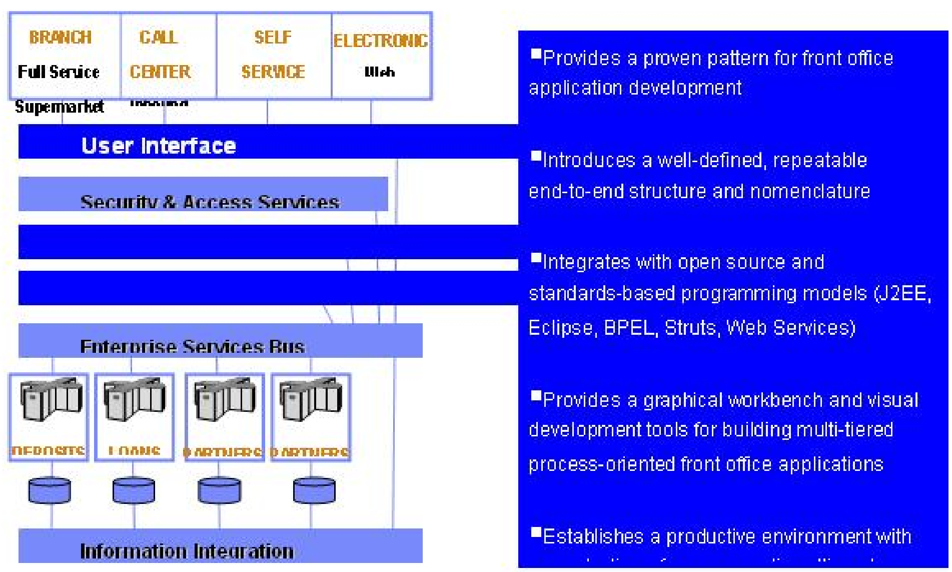
To pave the way for effective channel application investment, IBM provides a multichannel transformation based on a Service Oriented Architecture (SOA).

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
  - Mobile Banking

  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.

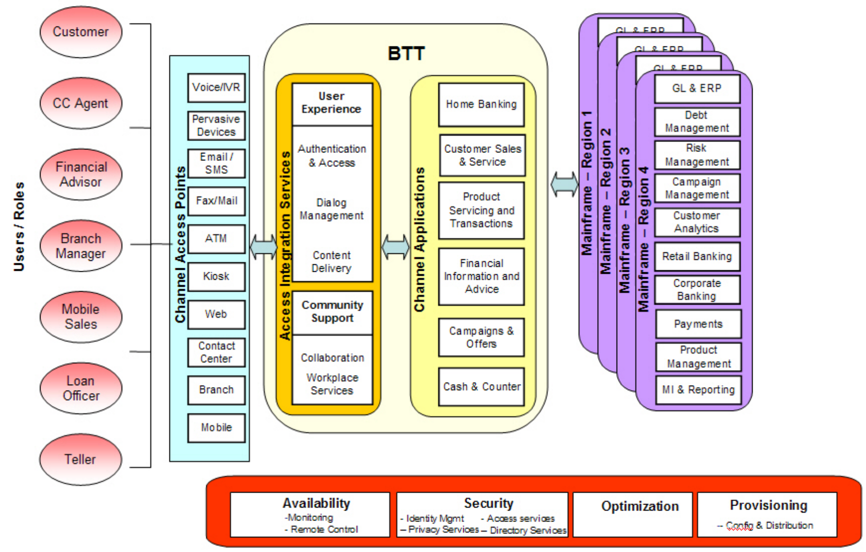
- **Supports replaceable components** - Components are packages of system function with established interfaces and a predetermined execution environment. As long as a component is within its required execution environment and it interacts with other system components through its public interfaces, it 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.

See the following figure:



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

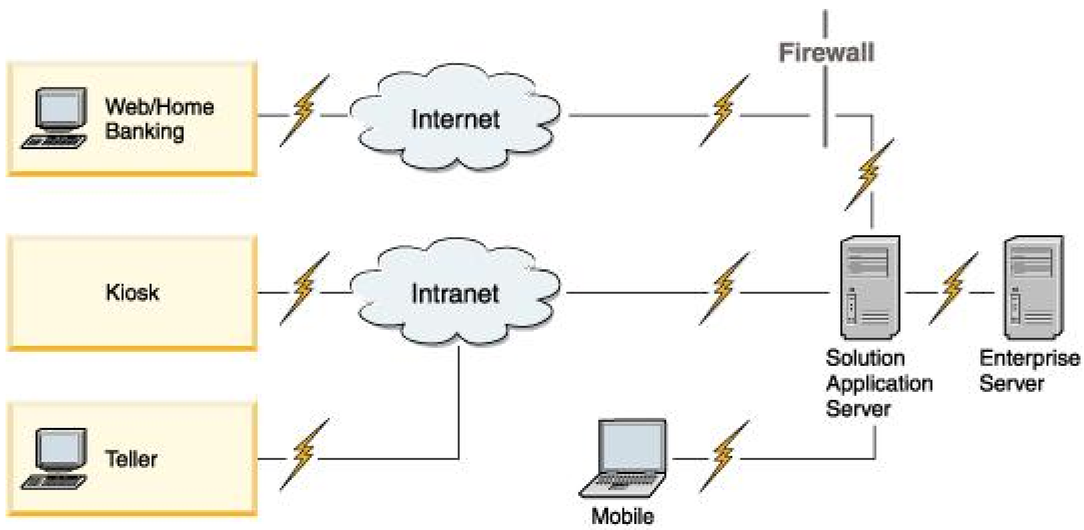
In an Internet banking topology, users obtain access to financial services through a Web browser (or other device) connected to the Internet. The user interface is normally HTML with additional technologies such as JavaScript, DHTML, or XML. In such an environment, the solution application server is 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

Figure 1 on page 10 shows the architecture of IBM WebSphere Multichannel Bank Transformation Toolkit (BTT):

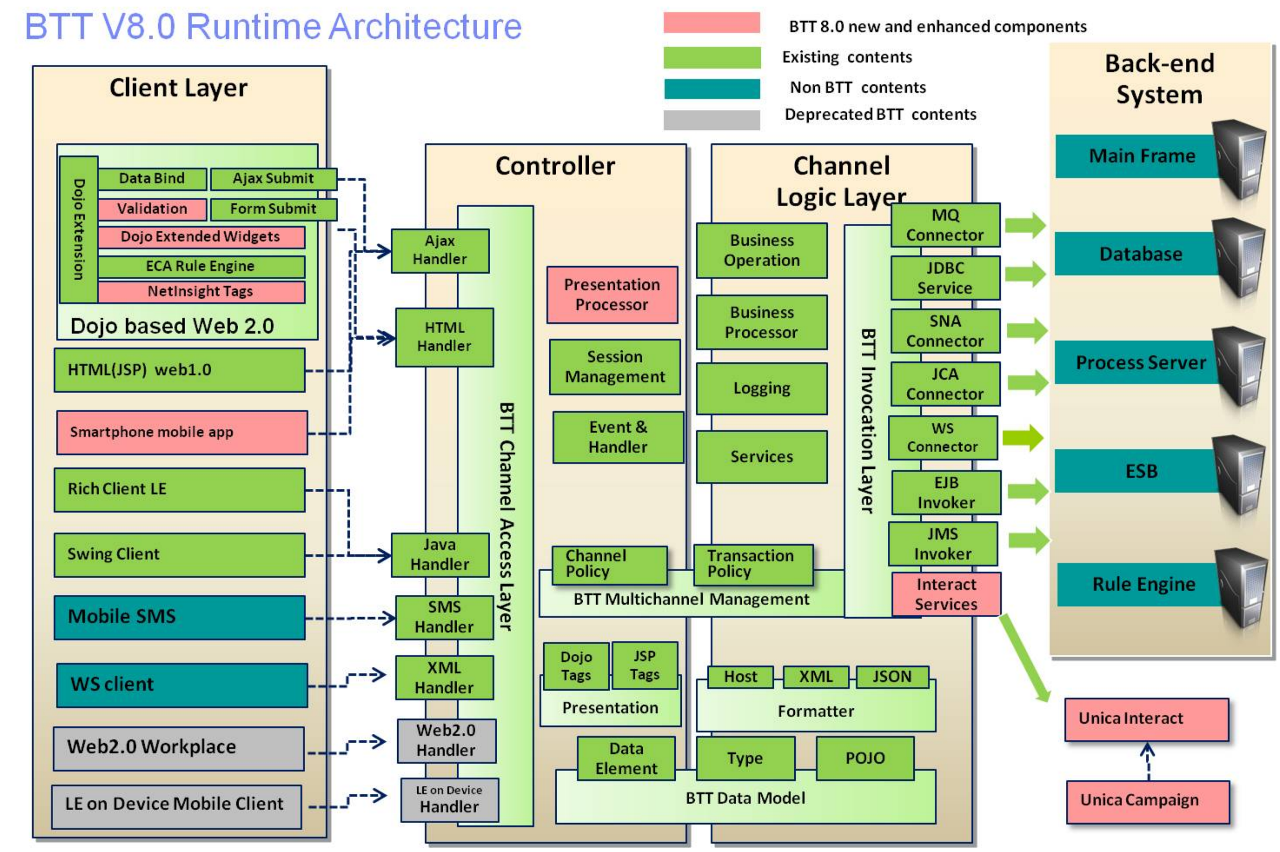


Figure 1. Architecture of WebSphere Multichannel Bank Transformation Toolkit (BTT)

Bank Transformation Toolkit best practices

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

Programmers' tips

This section contains a set of recommendations and programmers' tips 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 that 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 7.0, 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.

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.

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.

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.