IBM Branch Transformation Toolkit
for WebSphere Studio

Solution Architecture

Version 5.1

Note!
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 33.

Eighth Edition (July 2005)

This edition applies to Version 5, Release 1, Modification 0, of *IBM Branch Transformation Toolkit for WebSphere Studio* (5648-D89) 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
Information Development
10F Shui On Plaza, 333 Middle Huai Hai Road
Shanghai 200021, 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

Solution Architecture Overview

This document is mainly for Solution Architects, who require an overall description of what the IBM® Branch Transformation Toolkit for WebSphere® Studio (Branch 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 Branch Transformation Toolkit for WebSphere Studio is a component-based toolkit for developing enterprise e-business applications. The Branch 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 Branch 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 Branch 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, 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.

Architectural objectives

The architectural objectives of the IBM Branch Transformation Toolkit for WebSphere Studio 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 Branch 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 have 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 Branch 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.

Support for multiple channels

The IBM Branch Transformation Toolkit for WebSphere Studio 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 applicationon 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.

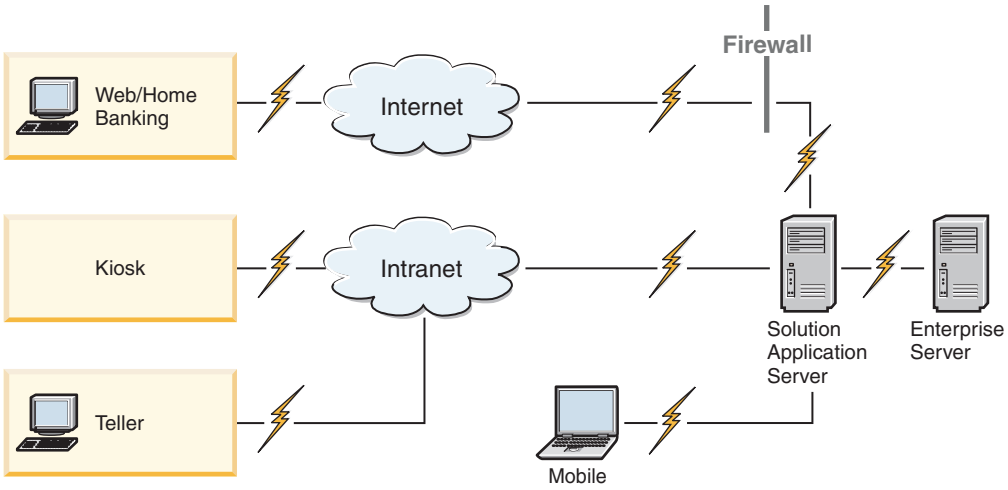
Kiosks

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
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:

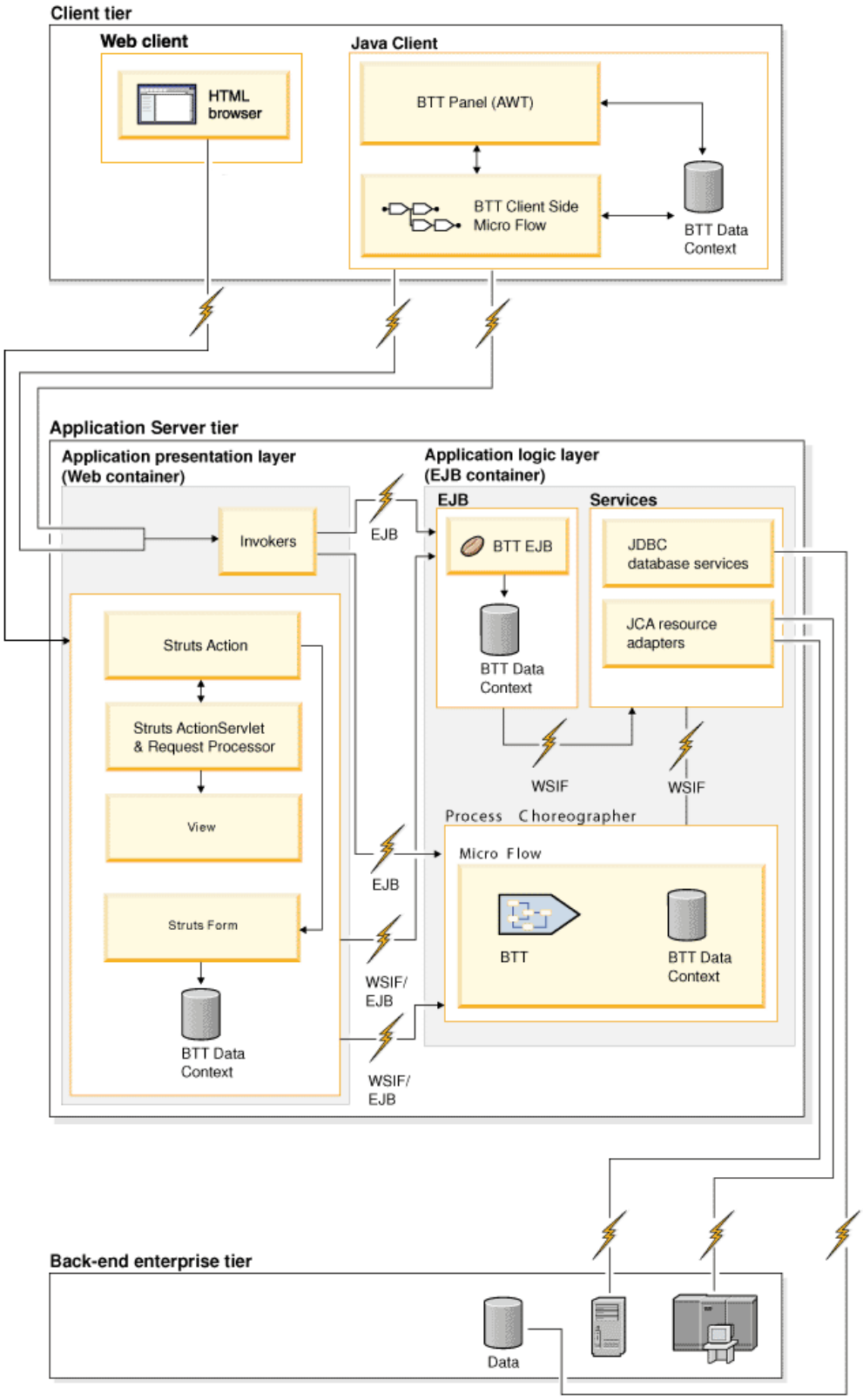


Architecture

The architecture of the Branch Transformation Toolkit application solution is based on a logical three-tier model: back-end enterprise tier, application server tier, and client tier.

Within the application server tier, the toolkit has two separate layers. The application presentation layer is responsible for receiving requests from the client and passing that request on to the application logic layer. It also passes the response back to the client. The application logic layer is responsible for performing the request as a process and passing the response back to the application presentation layer. The individual components within the layer are discussed later in this document.

For the most part, the application presentation layer resides in a Web container in WebSphere Application Server while the application logic layer resides in a EJB container. The services are the exception because they can reside anywhere.



The design and portability of the toolkit (resulting from being Java code) allow the middle-tier servers to exist at either the branch level (one server per branch), the regional level (one server per group of branches), or even a centralized level (a single server for the entire financial institution). The design provides flexibility to

achieve the right balance between the number of servers and the network bandwidth without affecting any application logic. Besides the application server, there may be a ″technical server″ responsible for providing common services (such as disks or printers) to the client workstations. If the application presentation layer and the application logic layer are on the same architectural level, they can physically be on the same machine.

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 access financial devices or validate 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 Branch Transformation Toolkit supports any kind of physical client device that uses the following technologies:

- Java applets in a browser environment
- Java applications
- HTML clients

The toolkit provides implementations for current client technologies but these concrete implementations anticipate that significant differences may 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.

A clear separation exists between Java clients and HTML clients. For a Java client, the application, which may also be executed inside a browser, can be built from toolkit-provided visual components (implemented as Java beans) using visual composition. The visual components of the toolkit and the interaction with toolkit services facilitate implementing required application tasks such as interacting with financial devices, database access, and other services. For an HTML client, the flow of the navigation is delegated to the server.

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 uses 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.

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.

Application logic layer

The application logic layer provides the core business logic using **Enterprise JavaBeans**™. It does this in a channel neutral manner. That is, it handles a transfer funds request whether the request came from a Web client or kiosk.

The mechanism for performing the business logic is a business process running in the Process Choreographer in WebSphere Business Integration Server Foundation or a business process running as a Single Action EJB. The business process can involve interacting with Web services, host applications using the JCA Connectors, and enterprise datasources to fulfill the request. The toolkit provides a set of services that support the application logic layer by providing connectivity to enterprise datastores or to legacy systems.

If the application presentation layer and the application logic layer are both running on WebSphere Business Integration Server Foundation, the presentation layer can use a work area to pass the session IDs to the application logic layer. Otherwise the application presentation layer includes the session ID with the data required to process the business request in the request message.

Runtime architecture examples

This section contains two examples that describe the flow of two transactions from end to end: one with a Java client with the application presentation and application logic layers running on WebSphere Business Integration Server Foundation and the other an HTML client with the application presentation and application logic layers running on WebSphere Application Server. The transaction is a customer search in which the user, a teller, enters the data for the customer search criteria on a view. The view displays a list of customers who meet the search criteria.

Java client environment

In this example, the Java client is within a browser that has the Java plug-in and starts the client application. This is one of many possible implementations for a Java client but it makes the example independent of the virtual machine provided by the browser. The startup applet launches the XML Desktop. Once the desktop is available, the navigation controller and the actions configured for the visual components control the view navigation and which business processes the user requests. The flow processor is an implementation of the Automaton (a state machine) that controls what happens when user's actions reach the server in the application presentation layer.

The application presentation layer and application logic layer run on WebSphere Business Integration Server Foundation so that the example can show how the application logic layer uses the work area and Process Choreographer features of that edition.

1. The user requests a customer search and provides the required input data:

a. The user clicks a desktop button to search for a particular customer's data.
b. The search button has an associated operation panel. The panel contains a set of entry fields for the search criteria and a list field.
c. The user enters the required data. The operation panel enables the OK button only when the user has typed values in all mandatory fields.
d. When the user clicks OK, the client creates the customer search client operation and creates a context for it. The client then chains it to an upper level context. The client operation may identify the parent or the toolkit may use the default context of the client/server session as the parent.
e. The client operation checks that the operation context contains the data needed to process the operation. This validation is a cross-field validation. If data is missing, the client operation may execute a local service or send a request to a remote server. The client operation unformats the data resulting from executing the service and places the unformatted data in the operation context.
f. The client sends the client operation to the server using the multichannel support component.

2. The application presentation layer sends the customer search request to the application logic layer.
   a. In the server, the servlet acting as the request handler receives the customer search operation.
   b. The request handler calls the Bean Invoker Factory to get the invoker for the customer search operation.
   c. The request handler uses a formatter to populate the request with data from the context.
3. The request handler places the session ID in the work area.
4. The invoker makes an EJB call to the to Business Process Component on the application logic layer to execute the customer search process.
5. The application logic layer executes the business process:
   a. The Business Process Component receives the request and retrieves the session ID from the work area. Note that a previous process (typically a logon process) has created the session and the session CHA context.
   b. The Business Process Component creates a CHA context to hold the process data and chains the process context to the session context.
   c. The Business Process Component performs the process using the Process Choreographer.
   d. The Process Choreographer performs the activities of the process such as performing a search in the customer database and logging the search in an electronic journal.
   e. The Business Process Component creates the response message and formats the data resulting from the search into the response message.
   f. The Business Process Component sends the response back the presentation server.
6. The client view displays a list of customers matching the search criteria:
   a. The flow processor unformats the response into the process context. The flow processor broadcasts an event so that the navigation controller is aware that the customer search process has completed and its data is available.
   b. The navigation controller updates the Customer Search panel with the response data (in this case, a list of customers that match the provided search criteria).

HTML client environment

An HTML client is generally used for a home banking application built to use the Branch 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. In this example, the toolkit Struts Extensions in the presentation server controls the navigation. The application presentation layer and application logic layer run on WebSphere Application Server so that the example can show how the application logic layer works when the work area and Process Choreographer are not available and uses Single Action EJBs to perform the logic.

1. The user requests a customer search and provides the required input data:
   a. The user clicks a customer search link in the HTML desktop. This user action sends a request to toolkit Struts Extensions.
   b. The toolkit Struts Extensions component in the application presentation layer starts the corresponding action (Struts action A) specified in the Struts configuration file. That action then returns the JSP page name. The presentation layer processes the JSP into an HTML page.
   c. In the client side, the browser displays the HTML page.
   d. The user enters the input data and clicks a Submit button. This sends the form data as an HTTP post request to the Action servlet of the Struts Extensions which in turn starts a Struts action (Struts action B) specified in the Struts configuration file. The request data contains a process ID as hidden fields along with the other input data.
   e. Struts Action B requests an invoker from the Bean Invoker Factory.
   f. The invoker formats the data into a process context.
   g. The invoker calls a method in a Single Action EJB to perform a customer search process in the application logic layer. Struts action B uses a formatter to add data from the context as a parameter of the method call. The method call also includes the session ID.
2. The application logic layer executes the business process:
   a. The Single Action EJB performs the logic contained in the invoked method.
   b. The Single Action EJB returns the response to the presentation server.
3. The client view displays a list of customers matching the search criteria:
   a. The invoker unformats the response into the process context.
   b. Struts action B points to a JSP that generates an HTML page with the response to the customer search request. The page contains a list of customers who match the search criteria.
   c. The client displays the HTML page.

EJB and Web services architectures

The Branch Transformation Toolkit provides both runtime and development architectures for building front-end solutions that access back-end enterprise applications and data. As part of the runtime architecture, the toolkit uses an application presentation layer with a client and server and an application logic layer. For the application logic layer, the business logic resides in a business process running in the Process Choreographer of WebSphere Business Integration Server Foundation or within a Single Action EJB. The business process or EJB can invoke Web services to perform business logic.

The Web services architecture defines a dynamic business-to-business programming model, where services are published, discovered, and accessed through standard definitions and interfaces. The toolkit integrates with this architecture and complies with both schemas to provide the following benefits:

- The toolkit application code can access any EJB or Web service
- The toolkit can build transactional-access EJBs or Web services, which are components that delegate transactional integrity to an external transaction monitor such as IMS™
- The toolkit can be extended and customized to create black-box EJBs or Web services
- The toolkit can make implementing a client front-end to an EJB or Web Service easier while providing the overall application with the ″n tier″ network computing architecture model based on the internal architecture of the toolkit's multichannel application presentation layer

However, the main benefit is that the integration preserves the flexibility of the toolkit while providing the benefits of the EJB and Web services architectures and a clean path to a components model for toolkit-based solutions. The application presentation layer can access the EJB or Web service using the EJB interface or WSIF interface respectively. From the point of view of the application presentation layer, they are black boxes.

Architectural considerations

This section contains the set of considerations to be taken into account when wrapping the Branch Transformation Toolkit logic into an EJB or Web service. These considerations provide the application architect or designer with an understanding of the product's facilities for wrapping toolkit logic. Keep the following in mind when constructing a solution using EJBs or Web Services:

- Depending on the solution being implemented with the toolkit, encapsulating the logic in EJBs or Web Services may not provide any benefit in terms of transactional integrity. This is particularly true when the back-end system fully provides integrity and the logic implemented in the middle tier is a pure passthrough. That is, from the client's point of view the middle tier is acting as the front door of the server logic.
- Irrespective of your model choice, the toolkit provides its full set of features to build your end-to-end solution. You are free to define the proper architecture for the application and then use the toolkit to fulfill the requirements based on the chosen architecture.
- Due to the internal implementation of the toolkit, having more than one EJB or Web Service implemented by different uncoordinated teams (such as different vendors) running in the same Java Virtual Machine and the same namespace may result in some coexistence problems. Since the use of different namespaces by different solutions based on the Branch Transformation Toolkit cannot be always ensured, it is better not to run EJBs or Web Services that have been implemented using the toolkit in the same JVM with the same namespace. This can be easily achieved during the deployment of the components, by assigning the different EJBs or Web Services to different containers (perhaps under different application server nodes).

Tiers and components

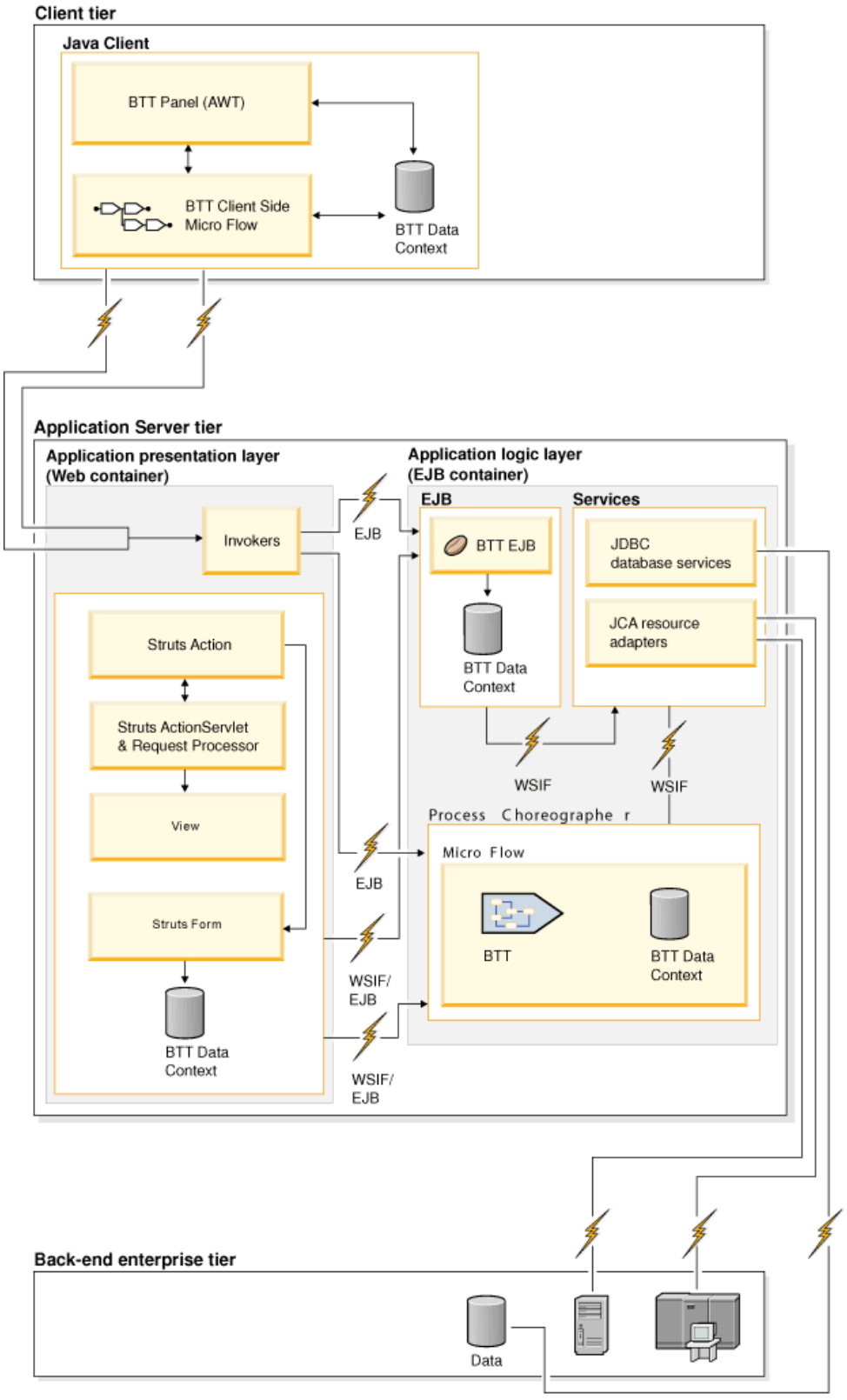
Branch Transformation Toolkit architecture consists of components, tiers or logical subsystems, subsystems, and Java packages, which can be used to define, view, and package solution technologies. The solution development process identifies the different parts of the architecture. For example, the requirements phase identifies components, the analysis phase identifies tiers and logical subsystems, and the design and coding phases identify physical subsystems and Java packages. Identifying these items during development ensures that the system structure is consistent and has integrity.

- **Components** are the building blocks of toolkit-based solutions. They are relatively independent and discrete parts that satisfy specific business or technical functions. Components have public interfaces that allow their functionality and implementation to evolve over time independent of the rest of the solution. To a certain degree, a toolkit-based solution is a group of reusable components, and the current deployed toolkit solutions are reference configurations of how to combine components to solve a technology problem for a customer channel.
- **Tiers (Logical subsystems)** are the analytical building blocks of toolkit-based solutions. They represent a partitioning of the system that is independent of the technology and physical implementation. The Branch Transformation Toolkit tiers map to the J2EE solution architecture.
- **Subsystems** represent its physical partitioning for deployment and execution (differing from tiers, which represent the logical partitions of a toolkit solution). A subsystem is a physically independent part of the system. As a result, it can be run on any computer within the system. A subsystem can be seen as a subset of components inside a domain.
- **Java packages** (also just called ″packages″) are the way that a toolkit-based solution delivers code. Packages provide a way of grouping functionality for a set of related classes. They define the namespace of a class, and follow a naming convention that commonly maps to domains and subsystems. The naming convention used by the Branch Transformation Toolkit is shown below:

  ```
  com.ibm.dse.domainName.subsystemName.furtherPackages.Class
  com.ibm.btt.domainName.subsystemName.furtherPackages.Class
  ```

  The ″tierName″ and ″subsystemName″ portions of the naming convention are determined by the system architect. The ″furtherPackages″ portion is for grouping classes into more discrete functional groups within a subsystem, and is determined by the subsystem designer.

The following chart shows how toolkit components and tiers map into the J2EE solution architecture.


Client tier
Java Client
BTT Panel (AWT)
BTT Client Side Micro Flow
BTT Data Context
Application Server tier
Application presentation layer (Web container)
Application logic layer (EJB container)
Invokers
EJB
EJB
Services
BTT EJB
BTT Data Context
JDBC database services
JCA resource adapters
Struts Action
Struts ActionServlet & Request Processor
View
Struts Form
BTT Data Context
WSIF
WSIF
Process Choreographe r
Micro Flow
BTT
BTT Data Context
EJB
WSIF/ EJB
WSIF/ EJB
Back-end enterprise tier
Data

Java client components in the client tier

The Java client components in the client tier of the J2EE architecture provides the entities for developing the Java client side of an application. Theses components control the user interface of the Java client, gather data from the user, sends requests to the application presentation tier, and receives response from the application presentation tier.

The toolkit architecture allows the externalization of most of these entities, thereby separating data for a specific client operation from the Java code. This reduces coding effort, which facilitates application development, enhancement, and maintenance.

Contexts, data elements, and typed data

A context groups data in a structure. Contexts are organized in a tree hierarchy and an application can define the context hierarchy statically but manipulate it dynamically at runtime. The contexts hierarchy groups data according to functional and business needs. For example, a context can group user-specific data, and a child of this context can hold operation-specific data. Contexts use a *chain of responsibility* pattern, so that if the toolkit requests some data element or service but does not find it in one context, it then searches in the parent of that context, and so on until it finds the data element or reaches the root context.

Data elements hold the values for the data or hold other data elements to form a hierarchy. Each data element is accessible by its name so that other components can retrieve or update the data. Data elements may be type-aware, which means that they can provide information about the type of the business object they represent. When the business object (for example, a Money object of type *Money* or a Customer object of type *Person*) is realized in Java code, the toolkit associates converters and validators with the object, handles the object cloning process, and holds any other business information required for the specific business object.

Formatters

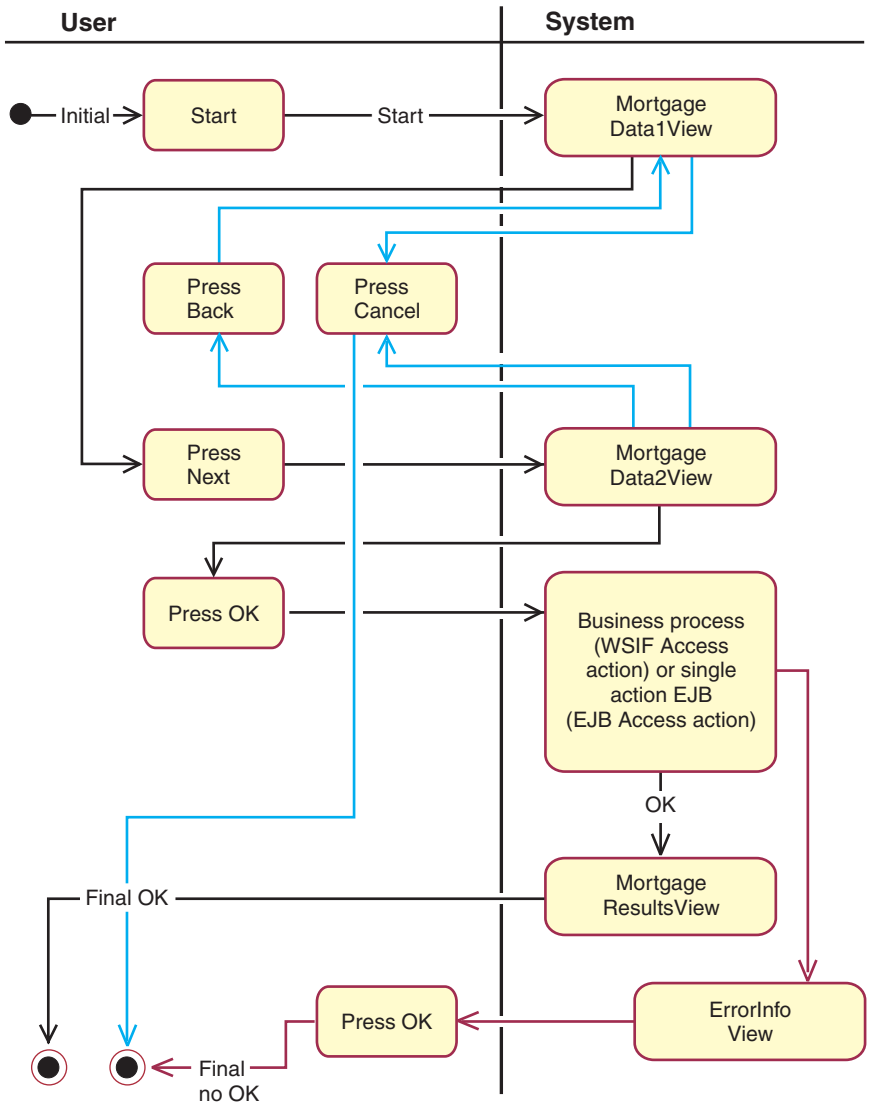
Formatters transform data elements into Strings and transform Strings into data elements. This enables the toolkit to send the data as String messages to and from a back-end host, or to a printer, and so on. In general, formatters format context data to present the data to services or to the application logic layer and to parse (unformat) messages received from services or the application logic layer to update context data. In some special cases, a formatter automatically maps between different namespaces.

The Branch Transformation Toolkit provides a rich hierarchy of Format classes to support a wide variety of message formatting requirements, and if the toolkit does not support some specific formatter, users can extend the hierarchy with their own subclasses.

Flow processors

A flow processor (an implementation of the Automaton, which is a state machine) defines a process in the application presentation layer as a series of states. In general, a flow processor performs an action in a state and transitions to another state based on the result of the action or based on events that occur within a view associated with the state. The flow processor can provide navigation from panel to panel, handle data mapping across the process, and launch client operations or business processes in response to certain events. This is demonstrated in the following statechart diagram for the flow processor that handles a simple mortgage

application. The boxes in the system side are the states and the boxes in the left are the choices available to the user.



To use a flow processor in this way, a user externally defines the set of states of the process, along with their corresponding transitions (specific events fired by the visual components) and the actions (such as perform a business process in the application logic layer, start another processor implementing a subflow, and open another view) for the flow processor. In this way, the user does not hardcode the execution of specific actions into the views.

Like the other components of the toolkit, users can extend all the constituent parts of a flow processor to get the exact behavior required in a specific customer environment. The toolkit provides entities as both interfaces and implemented classes with a base behavior that may be adequate in many cases.

Object factories (externalizers)

Externalizers are object factories that instantiate components from their definitions in external files, which specify their initial values and structure. To maximize code reuse, the toolkit is highly parametric. The externalizers read and process these parametric definitions. This process can generate a cascade of instantiations. For

example, when the user requests the instantiation of an operation from its external definition, the toolkit also instantiates its corresponding context and formatters using their externalizers. Instantiating the context implies instantiating its data and services and chaining it to its parent context. As a result, a small piece of Java code can make the externalizers build a complex structure.

Events

The toolkit provides a specialization of the JavaBeans event model for the application presentation layer. The sources for toolkit events are notifiers, and components that receive and dispatch those events are handlers. The Event Manager propagates events from their notifiers to their registered handlers regardless of their location. This means that events can propagate across the network and handlers can be aware of remotely generated events. Event Managers on each machine collaborate to propagate remote events.

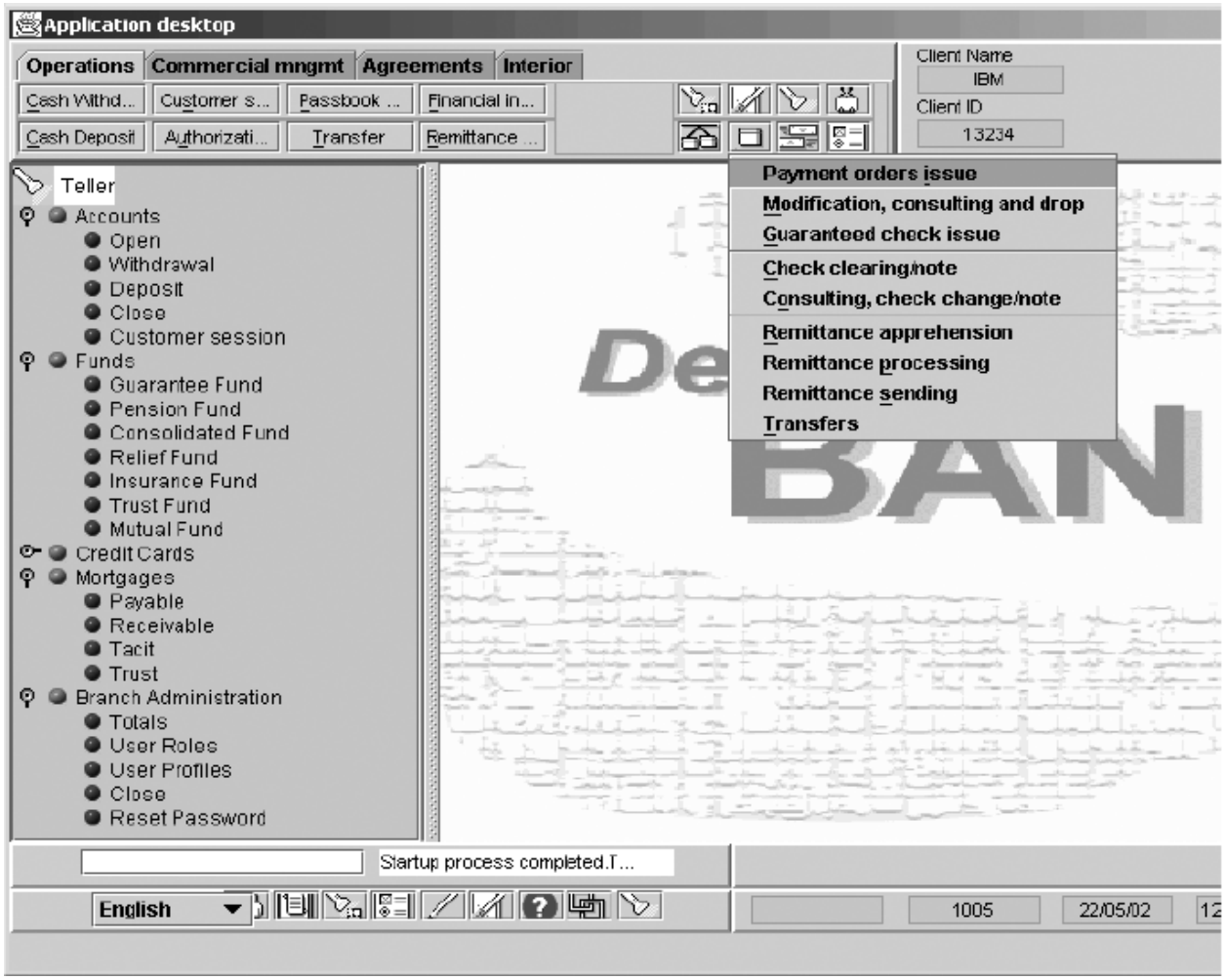
XML Desktop and visual beans

The Branch Transformation Toolkit provides two sets of visual components that allow the development of the graphical user interface (GUI) for a Java client. Both sets integrate with the underlying transaction posting infrastructure. These visual components are the XML (eXtensible Markup Language) Desktop and the GUI Beans.

XML Desktop

The XML Desktop component is a fully customizable desktop layout that provides the default behavior for interacting with the transaction processing of the toolkit infrastructure. It can be used as the entry point for an application, either as a standalone application, or as the main applet when started from a browser.

The XML Desktop implements reusable Desktop components that provide access to transactional processing and other features in a hierarchical manner and are easily added, removed, repositioned, and resized. The toolkit builds the desktop dynamically at runtime by adding the components defined in an external file that has the standard XML format. The following image shows a specific configuration of the XML Desktop that has most of the possible interactive elements enabled, such as Task Launcher Buttons, Icon Buttons, Menu Items, and LeafTree Nodes:



Some XML Desktop components implement the TaskLauncher interface, which the component uses to open a new task and display the initial panel for that task. The TaskLauncher associates the task with a single navigation flow, which steps the user through the different panels and executes the operations associated with the business process that the flow represents.

The TaskArea, which can resemble a task bar if desired, displays tasks. In the TaskArea, users switch from task to task by clicking on a button. The navigation controller assigned to the selected task handles navigation events resulting from the user's actions within the task's panels. The navigation controller is responsible for displaying the appropriate panel for each navigation event and for keeping track of previously displayed panels if the user navigates backwards.

The XML Desktop is fully localizable and can reference any locale-dependent resources (such as text and icons) from an external file. A LanguageBox desktop component is available so that users can change the active language at any time. If a user changes the active language, the toolkit regenerates the Desktop so that the text and icons are in the new language.

Visual beans

The Branch Transformation Toolkit provides a set of visual beans that:

- Extend the Swing visual beans behavior by adding characteristics that are common in financial applications
- Facilitate the integration of the data model of an operation with the view in a Model-View-Controller design pattern

Panel-related visual beans are an extension of Swing panels that update a given context when the user enters new data, generate navigation events that the navigation controller captures. They also refresh their own contents when the toolkit updates the context with data from a response.

The XML Desktop or any other view manager can manage views that users develop with visual beans. Formatters do field-level validation and format data for display. The toolkit provides a hierarchy of extensible formatter components and supports cross-field validation. It also supports displaying multiple errors simultaneously.

Trace Facility

The trace facility provides a class that keeps information in memory and records information on disk. The application or infrastructure classes can invoke the facility. Each entity calling the trace is free to trace any information it wants with some configurable settings to set the limits. The facility also provides a viewer tool, which allows the user to browse the trace entries that are in memory and make some dynamic changes to the tracing behavior.

The facility displays the trace information in a window and writes it to a file with a predefined maximum number of lines. When the file reaches its maximum number of lines, the facility returns to the beginning of the file and starts overwriting old information with the current information. This prevents the file from growing endlessly. The same consideration applies to the displaying window so that it does not use more and more memory while the code is being traced.

Traces are categorized following several independent criteria:

- By level: High, Medium, and Low
- By category: Information, Debug, Warning, Error, Information, etc.
- By component

Developers can enable or disable these categories depending on the tracing requirements for each situation.

Financial devices services

The Branch Transformation Toolkit provides services for accessing financial devices through two different standard APIs: WOSA/XFS and J/XFS. WOSA/XFS (Windows® Open Services Architecture / eXtension for Financial Services) is an industry standard based on C and Windows technology. The toolkit provides a wrapper that makes the WOSA/XFS API available to Java as a toolkit service. A toolkit-based solution can incorporate any device that has a WOSA/XFS device driver by defining the WOSA/XFS compliant device as an additional toolkit service. J/XFS (Java eXtensions for Financial Services) is an industry standard based on Java, which provides generic J/XFS Device Control objects as JavaBeans. The toolkit provides J/XFS Services by extending the J/XFS JavaBeans to provide the ability to define external parameters, to generate toolkit events, and to integrate with other toolkit components.

The toolkit also supports some financial devices by accessing them using the LANDP® protocol. The toolkit can issue from Java LANDP method calls, which enables a toolkit service to access any LANDP server.

Generic Pool

The Generic Pool service enables multiple application processes to share certain objects (classes and services) to make them reusable. This reuse reduces the average time to execute any process and also reduces the ″garbage collection″ work, since the system does not destroy the reusable objects after use. Unlike the financial device services, the Generic Pool service is also available in the application logic layer.

Components in the application server tier

The Branch Transformation Toolkit components that run on application servers can be divided into two categories: components running in the Web container of the application server and components running in the EJB container of the application server. The Web container and the EJB container share some toolkit components.

Shared components across containers

Some components are shared across the Web container and the EJB container. These components are needed both by the application presentation components for handling requests from clients and by the application logic components for doing business transactions. The shared components consists of:

Data elements and typed data
Same as those counterparts running in the Java client, data elements and typed data elements provides a mechanism for managing data used throughout a transaction.

CHA and CHA Formatter Service
The CHA and CHA Formatter Service support other components in the Web container and EJB container by providing a distributed data structure and mechanism to import and export data into the structure. The CHA uses the contexts, data elements, and typed data elements.

Events, externalizers, and exceptions
These components work in a same way with their counterparts running in the Java client.

Trace Facility
The trace facility provides a class that keeps information in memory and records information on disk. See “Trace Facility” on page 18 for details.

Application presentation components in the Web container

The application presentation components in the Web container provides the main entities for developing the application presentation layer part of an application. These components control the user interface of HTML clients, gather data from HTML clients, and launch the business processes performed in the application logic layer. This group includes the client/server connectivity components, which provide multichannel support connectivity between various client devices and the application presentation layer.

Invokers: Instantiated by the Bean Invoker Factory, invokers enables Struts actions or Java request handlers to access the business processes and Single Action EJBs through an EJB call.

When a request comes from a requester (the requester might be a request handler or a Struts action), the request brings a request ID and a session ID to the Bean

Invoker Factory. The request ID indicates what kind of transaction the client is requesting, and the session ID identifies the session of this transaction request. The Bean Invoker Factory generates or allocates an invoker with the request ID and session ID. The Bean Invoker Factory then returns the invoker to requester so that the requester can send the request to the application logic layer.

Java Client/Server Messaging APIs: The Java Client/Server Messaging APIs component is an implementation of multichannel support that allows the creation of distributed Java applications (not just distributed data but also distributed logic) using Internet technologies. The Java Client/Server Messaging APIs is based on the HTTP protocol but adds the concept of a session between the client and server. The Java Connector supports session clustering, load balancing, and a network dispatcher so that an application can be distributed among several servers.

Furthermore, the Java Client/Server Messaging APIs component allows a kind of dynamic application topology reconfiguration, which means that at any point in time the server executing the logic can be changed. A request from a toolkit client implies the execution of a business process or activity in the application logic layer. The request contains the name of the process or activity to execute, along with relevant data for unformatting into the process context, but does not specify where to execute the process or activity.

The Java Client/Server Messaging APIs component can use the SSL protocol capabilities, allowing secure information interchange between the client side and the server side of a toolkit-based system. The capability is particularly useful when the information is flowing through untrusted networks. The toolkit can use up to 1024-bit RSA for key exchange and 128-bit symmetric encryption of data.

Struts Extensions: The toolkit Struts Extensions enables HTML clients to send requests to and receive responses from a toolkit application using the HTTP protocol. The Struts Extensions provides additional functionality for handling the use of the back button in the browser, duplicate requests, NLS, validation, and error handling.

JSPs: JavaServer Pages (JSPs) are used to dynamically construct HTML pages requested by a client Web browser. Any HTML page can contain URL links that cause a JSP to be rendered and returned to the HTML browser. JSP tags render HTML contents based on the attributes of the tag and dynamic information obtained from the context.

Based on the Apache Struts Framework, the Branch Transformation Toolkit provides custom JSP tags and utility beans to enable applications to retrieve information from the context hierarchy, get resources, and handle errors. If your application requires additional behavior, you can build new tags using the StrutsJspContextServices interface.

Application logic components in the EJB container

The application logic components residing in the EJB container support the execution of business logic.

Business Process Component
The Business Process Component provides supporting entities so that an application can run a business process within the Process Choreographer of WebSphere Business Integration Server Foundation.

Single Action EJB
As an alternative to the Process Choreographer, an application can use a

Single Action EJB to perform the business process. A Single Action EJB may or may not use the CHA and CHA Formatter Service.

Startup beans
The application logic domain uses startup beans to initialize the application logic layer entities.

Communication services
Communication services provide connectivity to the existing data and applications in the enterprise systems. These services isolate the client application from the communications complexity by providing a clear and easy public interface. The SNA JCA LU0 Connector and the SNA JCA LU62 Connector are resource adapters that enable business processes

JDBC Database services
The JDBC Database services of the Branch Transformation Toolkit interact with a database through the JDBC protocol to provide access to database tables. The services also map context data to database records and database records to context data using a formatter. The Database Table Mapping service enables toolkit applications to access databases using a common interface. The Electric Journal service can record the services and processes used or performed by an entity such as branch, user, or terminal. The design of each application determines what information the service records as well as when it writes that information.

Generic Pool
The Generic Pool service in the application logic layer performs the same function as it does in the Java clients and Web Container.

Tools

The Branch Transformation Toolkit provides a number of tools that support the development of applications. All the tools are plug-ins of WebSphere Studio Application Developer or WebSphere Studio Application Developer Integration Edition. Note that some functions of the Graphical Builder are only available when you are using the Integration Edition of WebSphere Studio Application Developer.

The Graphical Builder provides a set of tools to define entities required by applications, and distribute the runtime files. The Graphical Builder provides a development environment throughout the development cycle of toolkit applications. It also acts as a portal from where you can start other tools that the toolkit provides.

The CHA Editor and Formatter Editor provide user-friendly interfaces for creating or maintaining the definitions needed by the applications in the application logic layer.

The Business Process BTT Wizard provides a graphical user interface (GUI) to help you extend your business processes for taking advantage of toolkit specific entities. Note that this tool is only available when you are using the Integration Edition of WebSphere Studio Application Developer.

The Struts Tools BTT Extension provides a GUI to help you extend your Struts configuration files for taking advantage of toolkit specific entities.

Apart from all the development tools listed above, the toolkit also provides a toolkit migration tool to help you migrate your toolkit applications developed with version 4.3 of the toolkit to the new version 5.1 architecture.

Development

The Branch Transformation Toolkit was developed using WebSphere Studio Application Developer or WebSphere Studio Application Developer Integration Edition. The Branch 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.

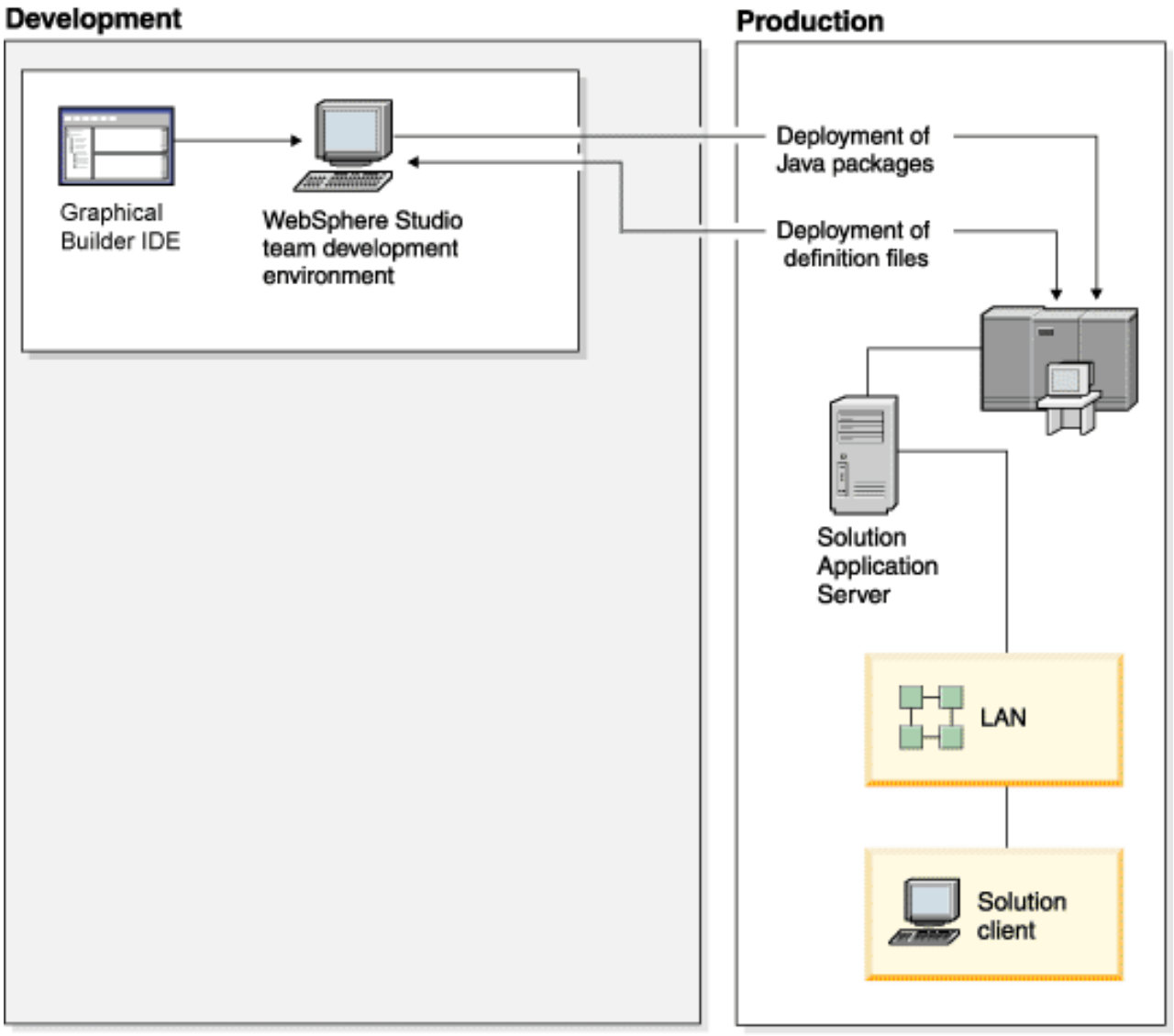
The following development tools may be used to build a solution using the Branch Transformation Toolkit:

- WebSphere Studio Application Developer or WebSphere Studio Application Developer Integration Edition to develop the application runtime engine and views, exploiting the required components provided by the toolkit using the following plug-ins:
  - The Graphical Builder to define the entities required by the applications in runtime in the application presentation layer, and help the user to more quickly deploy the set of resources needed by the runtime application when new business functions are added, all focused to encourage reusability of work, lower the costs of maintenance, distribution, and installation, and reduce the overall time to market. The Graphical Builder also acts as a portal from where you can start other toolkit provided tools.

    Note that certain functions of the Graphical Builder are only available when you are using the the Integration Edition of WebSphere Studio Application Developer.
  - The CHA Editor to define the CHA contexts and the data elements they contain.
  - The Format Editor to define formatters for the CHA contexts.
  - The Business Process BTT Wizard to extend your business processes for taking advantage of toolkit specific entities through a Graphical User Interface (GUI). Note that this tool is only available when you are using the Integration Edition of WebSphere Studio Application Developer.
  - The Struts Tools BTT Extension toextend your Struts configuration files for taking advantage of toolkit specific entities such as CHA contexts through a GUI.
- An authoring tool such as IBM WebSphere Studio Site Developer to create HTML pages.

Development Model

The Branch 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.

The Branch Transformation Toolkit provides development tools such as Graphical Builder to help you create, modify, and the definition files.

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 Branch Transformation Toolkit

development and where you can find more information on these tasks, see Creating an application with Branch Transformation Toolkit.

Physical deployment

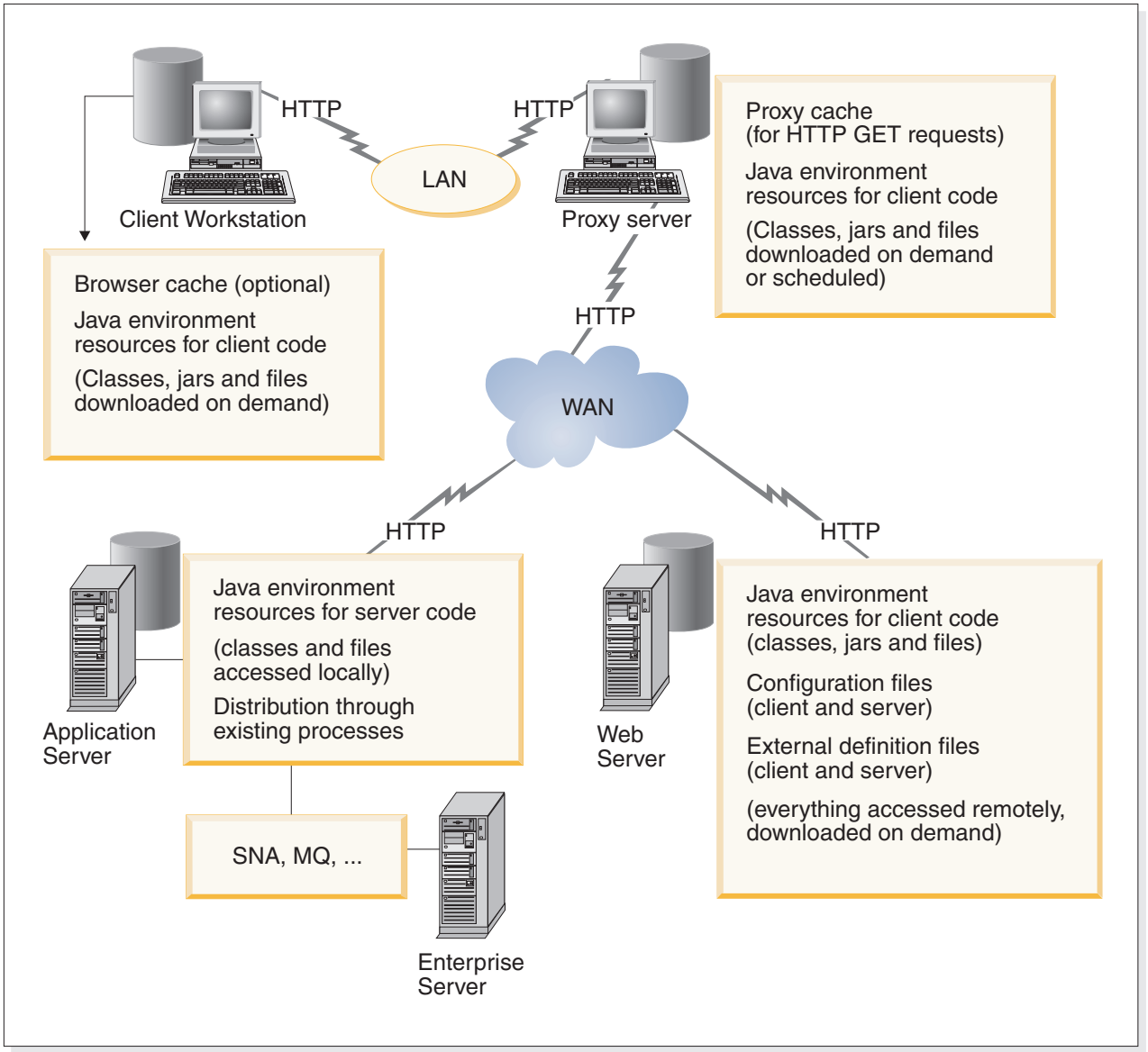
A toolkit-based application should use the standard mechanisms of the Internet or network computing technology for the distribution of objects and should exploit the cache mechanisms to get the best response times.

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:

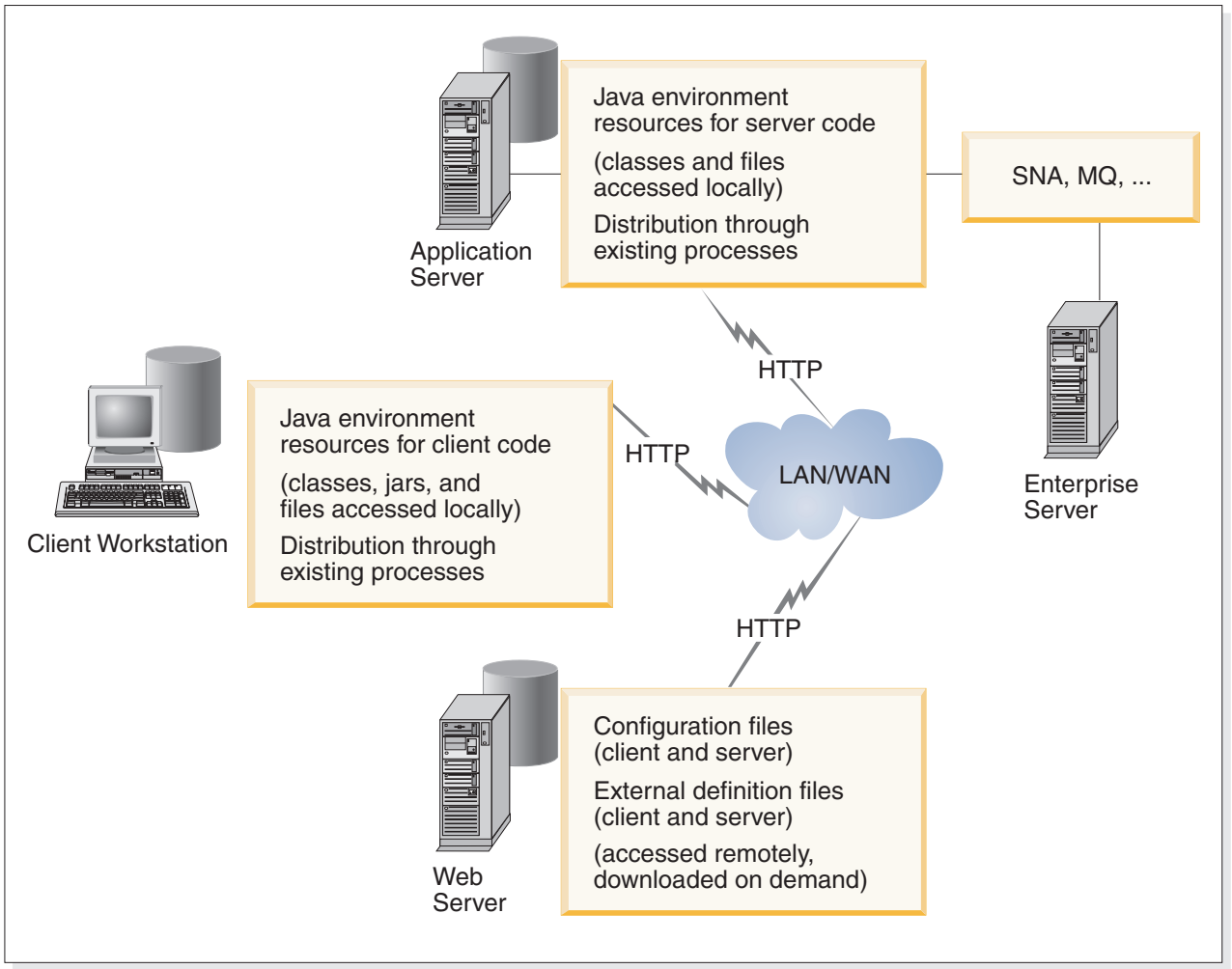
- The classes with their corresponding resources, obtained from the Java resources environment; that is, loaded by the existing class loader following the active classpath. This will be done regardless of whether the code is executed as an application (resources will be located locally) or as an applet 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 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 provide 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.

Performance tips

The performance tips given in this section are intended to help Branch 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 (check enableFormatsCache inside the initialization section). However, the application must exploit this feature by returning objects to the cache.
  - The HTML Connector returns all the mapper formatters to the cache. Only cacheable objects are cached.
- **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. An example is the "queueBufferSize" settingn at the main settings level, which defaults to 12. Also, consider removing any setting not required in your solution from the configuration file.
- **Shared typed data descriptors**:

  - With the addition of the parameters Hashtable into the base data element class, any data element instance may have its own parameters and reuse the same data descriptor instance associated with the unique data type, thus also reusing the type converters and validators. Check the ″shareDataDescriptors″ setting inside the initialization section of dse.ini.
- **Mapper formatters**:
  - When defining mappers to map elements from/to a flow to/from an operation or subflow, use DataMapperConverterFormat instead of DataMapperFormat. Also, consider using byReference=″true″ for each of the mapping elements (always keeping in mind the implications that this behavior may have in the context hierarchy).
- **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 much process 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.dse.automaton.ext and com.ibm.dse.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 instead of 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. If session persistence is enabled, tune the minimum boundary size of the session data to be persisted in order to enable compression (check setting minSizeForCompression inside the initialization section of DSE.INI).
- **Trace**:
  - Do not use the product trace mechanism as an application log. Instead, use database access services for this purpose.
  - Configure the trace settings according to the running environment. Settings such as whether to trace to file, intermediate buffer size (linesOfBuffer), and showOriginator have direct impact on the solution performance.
  - Use the Trace.doTrace method as a Boolean condition for tracing (before using the Trace.trace method) 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 doTrace method is true.
- **Application sessions table management**:
  - The application is responsible for maintaining the sessions table through the Context class protocol. It is crucial to clean session information from the table when the user logs off or when a session expires, to avoid apparent memory leaks and performance degradation. Both session entries and processor instances maintained in a session need to be removed from the table. Processor instances available in the cache also need to be removed in these cases.
- **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 Branch 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 depends 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 2000	Windows Server 2003	Windows XP	AIX®	Solaris	Linux® Intel®	z/OS®	Pure Java
Contexts, data elements, typed data elements		X	X	X	X	X	X		X
Formatters		X	X	X	X	X	X		X
Flow processors		X	X	X	X	X	X		X
Events		X	X	X	X	X	X		X
Externalizers		X	X	X	X	X	X		X
Exceptions		X	X	X	X	X	X		X
XML Desktop and Visual Beans		X	X	X			X		X
Financial devices	WOSA/XFS	X	X	X					
	J/XFS	X	X	X			X		X
	MSR/E LANDP	X	X				X		X
Generic pool		X	X	X	X	X	X		X

Table 2. Application server components

Component name		Windows 2000	Windows Server 2003	Windows XP	AIX	Solaris	Linux Intel	z/OS	Pure Java
Data elements and typed data elements		X	X	X	X	X	X	X	X
CHA		X	X	X	X	X	X	X	X
CHA Formatter Service		X	X	X	X	X	X	X	X
Java Client/Server Messaging APIs		X	X	X	X	X	X		X
Struts Extensions		X	X	X	X	X	X	X	X
Business Process Component		X	X	X	X	X	X	X	X
Single Action EJB		X	X	X	X	X	X	X	X
Startup beans		X	X	X	X	X	X	X	X
Events		X	X	X	X	X	X	X	X
Externalizers		X	X	X	X	X	X	X	X
Exceptions		X	X	X	X	X	X	X	X
Communications	JCA LU0 or LU62	X	X	X					
Financial devices	WOSA/XFS	X	X	X					
	J/XFS	X	X	X			X		X
	MSR/E LANDP	X	X				X		X

Table 2. Application server components (continued)

Component name		Windows 2000	Windows Server 2003	Windows XP	AIX	Solaris	Linux Intel	z/OS	Pure Java
Database services	Database Table Mapping	X	X	X	X	X	X	X	X
	Electronic Journal	X	X	X	X	X	X	X	X
Generic pool		X	X	X	X	X	X	X	X

Table 3. Tools

Component name	Windows 2000	Windows Server 2003	Windows XP	AIX	Solaris	Linux Intel	Pure Java
Graphical Builder	X	X	X			X	X
CHA Editor	X	X	X			X	X
Format Editor	X	X	X			X	X
Business Process BTT Wizard	X	X	X			X	X
Struts Tools BTT Extension	X	X	X			X	X

Components and technical requirements

The following table shows the additional technical prerequisites of the Branch 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
	MSR/E LANDP	LANDP, customized for device-specific servers
Database services	Table Access	JDBC driver
	Electronic Journal	JDBC driver
Tools		WebSphere Studio Application Developer or WebSphere Studio Application Developer Integration Edition JDK, JDBC Driver

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
2/F Deshi Building, No.9 Shangdi Dong Rd,
Beijing 100085, 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 and service marks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

IBM	OS/390
AIX	z/OS
CICS	LANDP
WebSphere	Tivoli
DB2	DB2 Universal Database
Informix	IMS
MQSeries	RS/6000
zSeries	

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

MMX, Pentium, and ProShare are trademarks or registered trademarks of Intel 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.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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