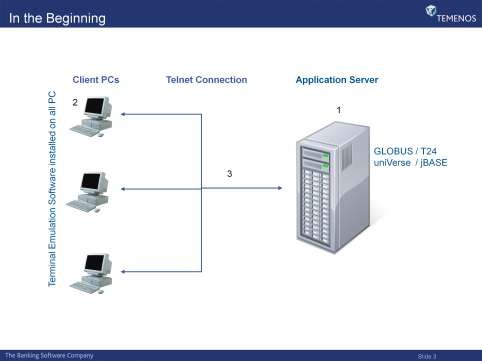
**T24 Architecture Evolution - Layman’s Explanation**

**1. Introduction to T24 Architecture Evolution**

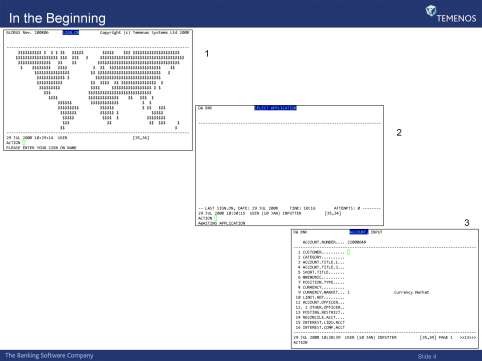
T24, a banking software system, has gone through multiple changes over the years in how it's built and how it works. These changes are mostly about making the system more **efficient**, **scalable**, and **easy to use**. We’ll explore how T24 evolved from a basic system to a complex, web-based solution that can handle millions of transactions smoothly.

**2. The Early Days (GLOBUS - The Beginning)**

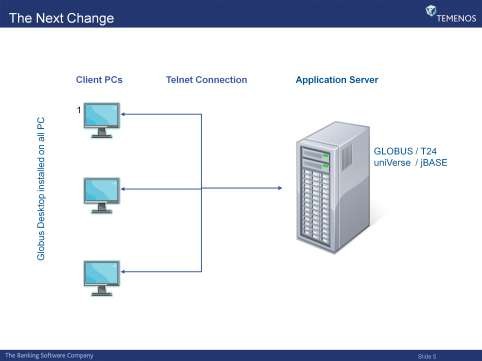


Imagine you're using a **traditional computer setup** where everything is stored in one place: your computer, applications, and data are all on the same machine. In the **early days**, T24 was known as **GLOBUS** and worked with a very basic setup:

* **Single Server Architecture**: Back then, everything (the application and the database) was stored on one server, just like putting your files, your word processor, and your internet browser all on the same computer.
* **Terminal Emulation**: Clients (bank employees using the system) didn’t have full software installed on their computers. Instead, they used a special tool called **terminal emulation software** (like Putty or Reflection). Think of it like a **remote control** for the server. The employees would use it to communicate with the GLOBUS server and access its applications, but all the heavy work was done on the server.
* **Text-based Interface**: When users logged in, they saw only **text on a screen** (like an old-school **command line interface**). There were no pictures, no buttons—just text that users typed commands into. It was functional, but not very user-friendly.



**3. Moving to Graphical Interface (GLOBUS Desktop - 1995)**



Now, imagine in the 90s when people started expecting **easier-to-use** software with buttons, pictures, and things that looked more like modern applications. T24 began to **upgrade**:

* **Graphical User Interface (GUI)**: GLOBUS transitioned to something called the **GLOBUS Desktop**. This allowed users to interact with T24 through a more modern-looking **Graphical User Interface (GUI)**. It was like upgrading from a **black-and-white TV** to a **color TV**. The users could see buttons, menus, and forms instead of just lines of text.
* **Still a Thin Client**: However, even with this upgrade, users still needed to install **GLOBUS Desktop software** on their own computers. It was **thin-client** software, meaning it only displayed information and let users interact with the system, but the **real work** (processing and storing data) still happened on the server.
* **Communication via Telnet**: Just like before, users connected to the server using a special **connection** called **telnet**. This was a way for computers to "talk" to each other over the internet or a local network.
* **Important Note**: From **Release R7** and beyond, **GLOBUS Desktop** was **discontinued**, and users were pushed to use **browsers** to access the system.

**4. The Big Change - T24 (Release R04 and Beyond)**

By the time T24 reached **Release R04**, things really started to change. It’s like when you move from an old **desktop computer** to the more **modern web-based software** we use today.

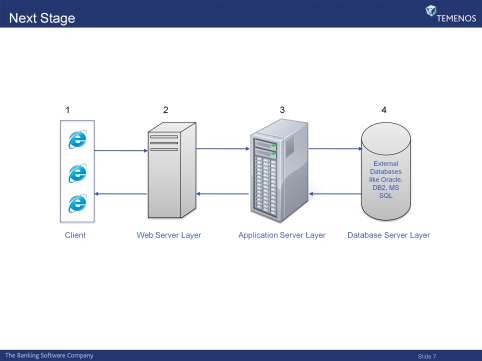
* **Web Browser Interface**: Instead of having to install software on every computer, T24 started using **web browsers** like **Internet Explorer** and **Mozilla Firefox**. This is the same way we access **websites** like Gmail or Facebook. All you need is a browser, and you’re good to go!
* **Rebranding from GLOBUS to T24**: Along with this, T24 decided to rebrand itself. **GLOBUS** became **T24**. The name **T24** stands for **Temenos 24** (meaning 24-hour, non-stop service), showing that the system was now capable of handling transactions all day, every day, with no downtime.
* **T24 Browser**: A new interface called **T24 Browser** was introduced. Initially, it was a **complementary solution** to the GLOBUS Desktop, but over time, it became the **main way** users would interact with T24. With **T24 Browser**, users didn’t need to install anything—they just opened a browser, typed in the address, and got started!

**5. The Modern T24 Multi-Tier Architecture**

Here’s where things get a little technical, but I’ll explain it simply. The new version of T24 is built using a **multi-tier architecture**.

* **What is a Multi-Tier Architecture?**
  + Imagine a **hotel** with multiple floors, and each floor has a different role. One floor might handle **reception**, another one deals with **guest services**, and another one is just for **rooms** (where guests stay).
  + In the same way, the **multi-tier architecture** in T24 has **multiple layers (tiers)**, where each layer is responsible for specific tasks. This makes T24 **scalable** (can handle more users or data) and **reliable** (if one layer fails, others keep things running).

Here’s how the T24 multi-tier system works:



1. **Client Layer** (The Front Desk):
   * **What it does**: This is what the user interacts with. It’s the **web browser** (like Google Chrome, Firefox, etc.).
   * **Example**: When you log into a website, you're using a **client** (browser) to interact with that website. In T24, the **client** is the browser where you access all the T24 functionalities.
2. **Web Server Layer** (The Reception Desk):
   * **What it does**: When you send a request (like trying to log in or open a new transaction), it first reaches the **Web Server**. Think of this as the **reception desk** of the hotel. The reception takes your request and forwards it to the right department.
   * **Example**: You want to check in, so you tell the receptionist your name. The receptionist then directs you to your room or gives you a key. In T24, the web server handles the request and sends it to the right server that processes it.
3. **T24 Application Server Layer** (The Back Office):
   * **What it does**: This is where the **real work** happens. The **Application Servers** are where the business logic (the actual banking processes) takes place. After the Web Server gets your request, it sends it here to be **processed**.
   * **Example**: Think of it like the **staff** working behind the scenes in the hotel, handling the complex requests and making sure everything runs smoothly.
4. **Database Server Layer** (The Storage Room):
   * **What it does**: This is where all the data (like user information, transactions, etc.) is stored.
   * **Example**: In a hotel, this would be the **storeroom**, where all guest information, inventory, and logs are kept safe and secure.

**6. Optional Components - Enhancements to Performance**

T24 also includes **optional components** that help improve the performance and reliability of the system:

* **Network Dispatcher**: This is like a **traffic manager** who ensures that requests coming from users are distributed across multiple servers, preventing one server from getting overloaded.
* **Message Queuing Software**: Think of this like a **post office**. When a user sends a request, it gets placed in a queue to be processed in order. If one part of the system is busy, the request waits until it can be handled.

**7. Example of T24 Screens**

Finally, here’s what you might see when using T24 through a browser:

1. **Sign-on Screen**: The login screen where you type in your username and password.
2. **Home Page**: Once logged in, you see your dashboard or home page, which might look different depending on your role.
3. **Application Screen**: After you click on a function, you’ll be taken to the screen where you perform tasks, like processing transactions or managing accounts.

**Conclusion**

In simple terms, T24 evolved from a **basic server-client model** (where everything was installed on a single server) to a **modern web-based system** where everything is distributed across multiple servers. The introduction of the **multi-tier architecture** helped make T24 more **reliable**, **scalable**, and **easier to use**. Now, users can access the system simply by opening a browser and logging in—no software installation is required.

**Introduction to the new architecture:-**



**1. JCA (Java Connector Architecture)**

**What it is**:

* **JCA** is a technology used in Java to connect **application servers** to different **information systems**, such as **relational databases** or other enterprise systems. It helps to establish a connection between Java-based applications and various external systems.
* You can think of it like a **universal adapter** that helps Java applications communicate with different systems, much like how different phone chargers can connect to various devices with the right adapters.

**How it works**:

* JCA allows Java applications to use standard methods to **connect to enterprise systems**, such as databases or messaging systems. It simplifies the process of connecting to and interacting with these systems.

**Why it's important for T24**:

* JCA is an alternative to older, proprietary methods used to connect to systems like TCServer. It’s more **flexible**, as it doesn't rely on a single, proprietary connection method.

**2. Application Server**

**What it is**:

* An **Application Server** is software that hosts applications and provides the environment to run them. It is primarily used for **serving business logic** (the core processing part of applications) and handling complex backend functions.
* You can think of it as a **server** that delivers and manages the business services, like a **hotel manager** organizing everything from guest check-ins to housekeeping.

**Key features**:

* **Deployment environment**: The application server is where the business logic runs, and it coordinates updates to the data.
* **Web Container**: Hosts web-based applications (like a website).
* **EJB Container**: Hosts **Enterprise JavaBeans**, which handle backend processes in enterprise applications.

**Popular examples**:

* **JBoss**, **Weblogic**, and **WebSphere** are common application servers used in many enterprise applications, including T24.

**3. Resource Adapter**

**What it is**:

* A **Resource Adapter** is a software component that allows an **external system** (like a database or messaging system) to connect to an **application server**.
* You can think of it like a **plug** that connects external systems to your network or application. It ensures that everything communicates smoothly and follows the right protocols.

**How it works**:

* A **Resource Adapter** follows **JCA standards** to integrate external systems with Java-based applications. It’s like a **driver** that helps the application server interact with other enterprise systems.
* Example: A **JDBC driver** that connects Java applications to a database is a type of resource adapter.

**Why it’s important for T24**:

* **Resource adapters** ensure that T24 can connect to various external enterprise systems efficiently without custom configurations.

**4. Enterprise Java Beans (EJB)**

**What it is**:

* **EJB** (Enterprise JavaBeans) is a **server-side component architecture** for Java applications. It’s a way of organizing and running business logic in a distributed environment.
* You can think of **EJBs** as **software modules** that handle important tasks (like processing customer data or transactions) in an enterprise application.

**Types of EJB**:

* **Session Beans**: These handle the business logic of an application. They can be thought of like **workers** who perform tasks on behalf of users.
* **Entity Beans**: These represent **data objects**, like customer records or transactions, that are stored in a database.
* **Message Driven Beans (MDB)**: These handle **asynchronous messages** (messages that don't need an immediate response), helping with communication between different parts of the system.

**How it works**:

* EJBs are managed by an **EJB container** (like an **application server**). These beans provide a **standardized way** of handling backend business logic and ensure that the application is scalable and manageable.

**5. Message Driven Beans (MDB)**

**What it is**:

* **Message Driven Beans (MDB)** are special types of **Enterprise JavaBeans (EJBs)** that process **asynchronous messages** in Java applications.
* Imagine you send a letter to a company and don’t expect an immediate reply. An MDB processes such **messages** in the background, like a **post office** that handles packages and letters without needing to respond right away.

**How it works**:

* MDBs listen for incoming messages (like **emails** or **notifications**) and process them when they arrive. This makes them useful for handling tasks like updating databases or triggering events based on received data.

**Why it's important for T24**:

* MDBs help T24 handle background communication tasks efficiently, which is especially useful for **banking transactions**, where different parts of the system need to work together asynchronously (not all at once).

**6. Java Naming and Directory Interface (JNDI)**

**What it is**:

* **JNDI** is an API (a set of rules and protocols) that provides **naming and directory services** in Java applications. It allows applications to look up and find various resources, like **databases**, **files**, or **other devices**, on the network using simple names instead of complicated addresses.
* You can think of **JNDI** as a **phone book** where you can look up someone’s contact info using their name, rather than knowing their exact phone number or address.

**How it works**:

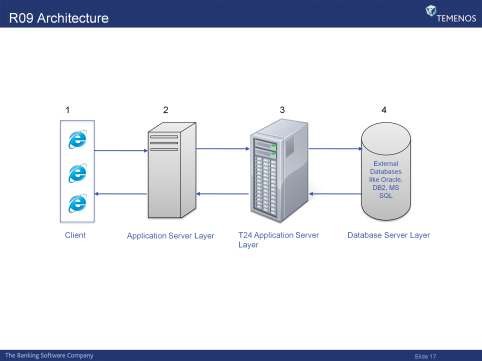
* JNDI helps Java applications find resources and objects like databases or messaging systems by searching a **directory**. This process makes it much easier for Java applications to **locate services** across different networks or systems.

**Why it’s important for T24**:

* T24 uses JNDI to manage how it connects to various **enterprise systems**, ensuring that connections to databases and other systems are simple and scalable.

**Summary**

These new concepts and components are part of the **modernized T24 architecture**, which is built on **Java technologies**. The introduction of technologies like **JCA**, **EJB**, **Resource Adapters**, and **JNDI** enables T24 to be **more flexible**, **scalable**, and capable of **integrating** seamlessly with other enterprise systems (like databases or messaging platforms).



Here’s a detailed explanation of the changes introduced in **T24 R09 architecture** and how each layer operates, with a focus on the **client-server relationship**, **application server**, **TAFC/TAFJ**, and the **two architectures**: **Agent Architecture** and **JMS Architecture**.

**1. Client Layer (T24 Browser compliant browser)**

**Overview**:

* To access **T24**, any **T24 Browser compliant browser** like **Internet Explorer** or **Mozilla Firefox** can be used as the **client software**.
* This browser must be installed on all client PCs in your network. Users can interact with T24 by accessing it through their web browser.

**Key Points**:

* This is a **web-based** access model where users don't need to install any thick client software but can just use a **web browser**.
* It brings **flexibility** since users can access the system from any machine that supports a web browser.

**2. Application Server Layer**

**Why Do We Need an Application Server?**

* An **application server** plays a crucial role in T24's architecture. While a **web server** simply receives HTTP requests and responds with HTTP responses, it lacks certain features necessary for business applications.

**Web Server vs Application Server**:

* **Web Server**:
  + **Simple delegation**: A web server only forwards requests to the application best suited to handle them.
  + **No transaction support**: It doesn’t handle transactions or manage database connections.
* **Application Server**:
  + Provides **business logic** for client applications.
  + Offers **additional features** like **load balancing**, **deployment APIs**, **connection pooling**, and more.
  + **Transaction management**: Ensures that requests are processed reliably and in an organized manner.
  + **Database connection pooling**: Helps manage connections to the database more efficiently.

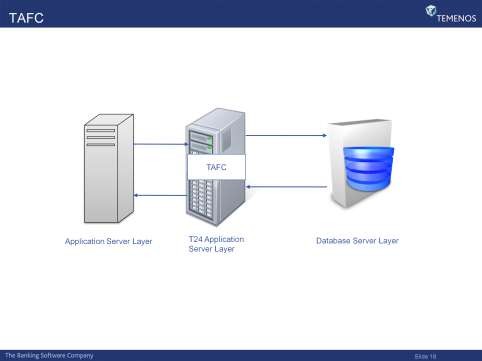
**Why is this needed for T24?**:

* T24 requires **more complex handling of requests**, transactions, and resources, which a web server cannot provide alone. This is why T24 uses **application servers**.

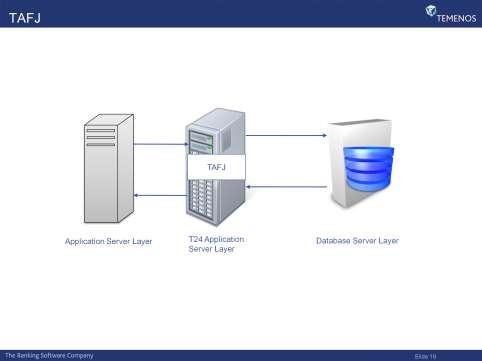
**3. T24 Application Server Layer (TAFC and TAFJ)**

**Types of T24 Application Servers**:

* **TAFC**: This is the **jBASE runtime** environment used by the T24 application server. **TAFC** is the server that processes requests from clients and handles business logic using the jBASE database.



* **TAFJ**: This is the **Java runtime environment** for T24. TAFJ processes requests using **Java-based components** rather than jBASE.



**4. Database Server Layer**

**Database Independence**:

* The **Database Server Layer** is where all the data for T24 is stored.
* **T24 is database independent**, meaning it can work with different types of databases (not restricted to a single database system like jBASE).

**How this fits into the architecture**:

* The **T24 Application Servers (TAFC/TAFJ)** interact with the database server to fetch or update data as required by business operations.

**Two architectures introduced with GA R09**

IT refers to the introduction of two distinct architectural approaches or configurations in **Temenos T24** with the **General Availability (GA) release version R09.00**.

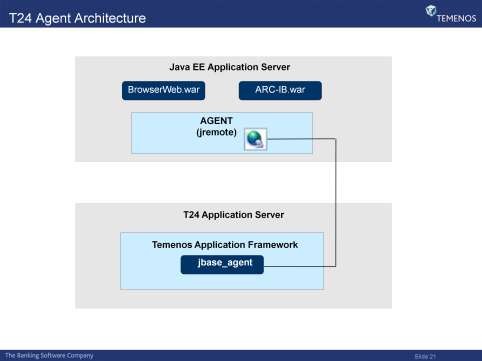
**Breaking it down:**

* **GA R09**: This is the General Availability release of **T24** version **R09.00**. "General Availability" means that this version of the software is officially released and available to customers for production use.
* **Two Architectures**: This means that Temenos has introduced **two different ways** of configuring or structuring the system architecture starting from the **R09 release**. These two architectures serve different needs or improve on certain features.

**The Two Architectures:**

The **Agent Architecture**

It was introduced as part of T24's transition to the R09 architecture. Here’s how it works:



**How the Agent Architecture Works**:

* The **T24 Browser** (client) communicates directly with T24 using the **TAFC JCA** (called **jremote** in this case).
* The **jbase\_agent** is used to facilitate communication between the client and the server.

**Key Components**:

* **jremote**:
  + **Client-side component** of jBASE that sends requests from the browser to T24.
  + It acts as the **interface** between the T24 client and the backend servers.
* **jbase\_agent**:
  + **Server-side component** of jBASE that handles the requests coming from **jremote**.
  + It listens for incoming requests via **TCP socket connections** and processes the client requests.
  + Think of **jbase\_agent** as a **server-side application** that helps the T24 system understand and respond to client requests from the **web browser**.

**6. Key Features of T24 R09 Architecture**

**Stateless Connection**:

* A **stateless connection** means that there is **no persistent connection** between the client and the server across multiple interactions.
  + Each request sent from the client is independent of the previous one, which makes the system more efficient and scalable.
  + **No acknowledgment**: The client sends a request and doesn’t need an acknowledgment or wait for one.

**Resilience**:

* **Resilience** means that the system is capable of **handling faults** and maintaining **acceptable service levels** even when there are failures.
  + Since multiple servers exist in each layer of T24, there is **no single point of failure**.
  + **Failover mechanisms**: In case one server fails, another takes its place seamlessly, ensuring continuous operation.

**Scalability**:

* **Scalability** means that T24 can handle **growing amounts of work** without performance degradation.
  + T24 can **scale up** efficiently by **adding more servers** at any point, allowing the system to handle **increased demand** as business grows.

**Summary of T24 R09 Architecture**

The **R09 architecture** brings several key improvements to the way T24 operates:

* The introduction of a **web browser interface** allows clients to access T24 from anywhere using a browser, without needing dedicated software.
* **Application servers** take over the heavy lifting of business logic, providing features like load balancing and transaction management.
* **TAFC** and **TAFJ** are the two core application server components, based on **jBASE** and **Java**, respectively.
* **Agent Architecture** introduces direct communication between the client (browser) and server (via **jremote** and **jbase\_agent**).
* **Scalability** and **resilience** are enhanced, allowing T24 to efficiently handle growing business demands and ensure continuous service.

This architecture significantly improves the **flexibility, reliability**, and **performance** of T24 as a core banking system, providing a better experience for users and administrators alike.

Here’s an overview of the **different layers** in the **Agent Architecture** for T24, breaking it down into manageable chunks for clarity:

**1. Client Layer**

* **Web Browser (1.1)**:
  + The **client interface** for accessing T24 is a **web browser** (e.g., **Internet Explorer** or **Mozilla Firefox**).
  + The **web browser** forms the **topmost layer** in the T24 architecture.
  + The client only needs to enter the T24 **URL**, which makes it **easy to deploy** since no special software needs to be installed. Most operating systems come with a browser pre-installed.
  + **Ease of Deployment**: As the client software is a browser, there’s minimal setup required on the user's machine.

**2. Application Server Layer**

The **Application Server Layer** in T24 consists of three key components:

* **Application Server Software**
* **T24 Browser Servlet**
* **jremote** (The resource adapter)

**Application Server:**

* An **application server** hosts the business logic and processes for third-party applications to interact with.
* It is responsible for **transaction management**, **load balancing**, **connection pooling**, and other enterprise-level functionalities.
* Popular examples include **JBoss**, **WebSphere**, and **WebLogic**.

**T24 Browser Servlet:**

* The **T24 Browser Servlet** is packaged as a .war (Web Archive) file and contains the logic for transforming incoming **HTTP requests** into a **Browser XML request**.
* Once the request is transformed, it forwards the request to **jremote** for further processing.

**jremote:**

* **jremote** is a **resource adapter** that allows the **T24 Browser** to communicate with the **T24 Application Server**.
* It forwards the request received from the **Browser Servlet** to the **T24 Application Server**.
* **jremote** is essentially the **communication bridge** between the **T24 Browser** and **T24 Application Server**.

**3. T24 Application Server Layer**

The **T24 Application Server Layer** has multiple components responsible for handling and processing requests and business logic:

**jbase\_agent:**

* **jbase\_agent** is the server-side component of jBASE. It is responsible for **accepting and processing** incoming client requests.
* It runs as a process on the T24 server and uses **TCP socket connections** to communicate with clients, meaning it listens on a user-defined **TCP port** to handle multiple client applications.
* Once a request is forwarded from **jremote**, **jbase\_agent** processes the request and hands it over to T24 for processing.

**T24 Runtime (TAFC):**

* **TAFC (Temenos Application Framework)** is the **jBASE runtime environment**.
* It provides the **business logic layer** where requests are processed, and it acts as the primary runtime for the T24 system.

**Direct Connect Drivers:**

* **Direct Connect Drivers** are used to connect **T24** to **external databases** like **Oracle**, **DB2**, or **SQL Server**.
* These drivers convert **T24 requests** into **SQL/DML/DDL** statements and send them to the external database.
* The response is then converted back to the T24-specific format and returned.
* These drivers contain database connection details like **username**, **password**, and **schema name** and must be installed on each of the T24 application servers.

**4. Database Server Layer**

T24 is **database-independent**, which means it can use different database systems to store its data. The **Database Server Layer** is responsible for storing and retrieving T24 data.

**Supported Databases:**

* **Oracle**, **DB2**, **SQL Server**, or **jBASE** can be used as the database for T24.
* **jBASE** is the default database for T24 but external databases can also be used with the appropriate drivers.

**Clustering:**

* **Database Clustering**: When using databases like **Oracle** or **DB2**, **clustering** is supported, meaning multiple copies of the same database are maintained for **resilience**. If one fails, another copy takes over.
* **jBASE** (prior to version 1.5) **does not support clustering**, but later versions do.

**Message Formats in T24:**

T24 can handle three types of message formats:

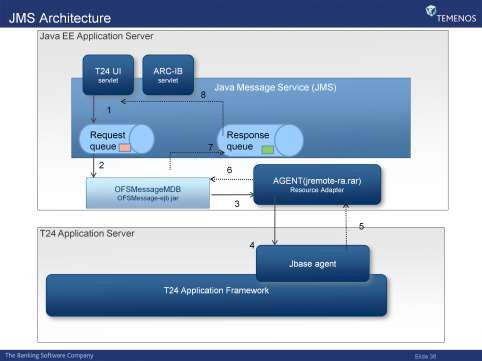
1. **Browser XML** – Used only when accessing T24 through the **T24 Browser**.
2. **OFSML** – Published format, used by **external systems** to interact with T24.
3. **OFS** – Native T24 message format, used for internal T24 communication.

**Flow of Requests in T24 Agent Architecture:**

1. **From the Browser to Application Server**:
   * The client (web browser) sends an **HTTP request**.
   * The **T24 Browser Servlet** transforms the HTTP request into **Browser XML** format.
   * The **Browser XML** request is forwarded to **jremote**, which in turn sends it to **jbase\_agent** on the T24 Application Server.
2. **Processing the Request**:
   * **jbase\_agent** sends the request to **T24** for processing.
   * T24 processes the request and builds a **Browser XML response**.
3. **Returning the Response**:
   * The **Browser XML response** is passed from **T24** to **jbase\_agent**.
   * The **jbase\_agent** sends the response back to **jremote**, which then sends it to the **T24 Browser**.
   * The **T24 Browser** converts the **Browser XML response** back to a standard **HTTP response** and sends it to the client.

**JMS Architecture**

**7. JMS Architecture in T24 R09**



Along with **Agent Architecture**, T24 also introduced the **JMS (Java Message Service) Architecture** in R09. This is another **communication model** for handling asynchronous messaging between different parts of the system.

**JMS Architecture** allows T24 to handle **message-based communication** more efficiently, especially for tasks like:

* **Handling notifications**.
* **Processing messages asynchronously**.

The key idea with **JMS** is that components of the system can **communicate by sending messages** to each other rather than making direct requests and waiting for immediate responses. This is particularly useful for large-scale systems.

**JMS** (Java Message Service) is an extension provided by J2EE application servers (like **JBoss**, **WebSphere**, and **WebLogic**) that adds support for reliable and asynchronous message processing.

**Components of JMS Architecture:**

1. **T24 BrowserServlet** – The front-end component that handles initial HTTP requests.
2. **OFSMessageMDB** – A **Message-Driven Bean (MDB)** that acts as a gateway for the T24 Browser to communicate via messages.
3. **TAFC JCA (jremote)** – The Java Connector Architecture (JCA) that forwards messages to T24.

**JMS Flow:**

* The **OFSMessageMDB** listens for messages from a **JMS queue** and processes them.
* Once a message is received, it is forwarded to T24 via **jremote** (the TAFC JCA component), which processes the request and sends a response back through the same process.

**Summary of the Agent Architecture:**

* **Client Layer**: The **web browser** allows clients to interact with T24 easily.
* **Application Server Layer**: Consists of an **application server**, **T24 Browser Servlet**, and **jremote** to handle requests and business logic.
* **T24 Application Server**: Components like **jbase\_agent** and **TAFC** process requests and interact with external databases via **Direct Connect Drivers**.
* **Database Server Layer**: Stores all T24 data in a database like **Oracle**, **DB2**, or **jBASE**, supporting clustering for resilience.
* **JMS Architecture**: Introduces **reliable message processing** for enhanced resilience and asynchronous communication.

Let's break down how requests are processed in the **JMS Architecture** for T24 and understand the components involved in the process. I'll also cover the new features and components like **TOCF Plugin** and **Network Dispatcher**.

**JMS Request Flow in T24**

Here’s a step-by-step breakdown of how the requests are processed in the **JMS architecture**:

1. **Request Dropped onto the Request Queue**:
   * The **T24 BrowserServlet** (or any external interface) drops a request onto the **request queue**.
   * The request could come from a **browser client** or from other **external interfaces** that need to interact with T24.
2. **OFSMessageMDB Listens and Picks Up the Request**:
   * **OFSMessageMDB** (Message-Driven Bean) acts as a listener on the request queue.
   * As soon as the request is dropped onto the queue, **OFSMessageMDB** is triggered to pick up the request for processing.
3. **Forwarding the Request to jremote**:
   * In order to process the request, **OFSMessageMDB** needs to forward it to the **T24 application server**.
   * To achieve this, **OFSMessageMDB** forwards the request to **jremote** (also known as **TAFC JCA**), as **jremote** is the component that allows communication with the T24 Application Server.
4. **jremote Communicates with the T24 Application Server**:
   * **jremote** then communicates with the **T24 Application Server** through **jbase\_agent**.
   * **jbase\_agent** is a server-side component that processes the request and forwards it to **T24** for business logic execution.
5. **T24 Processes the Request and Responds**:
   * Once **T24** processes the request, it prepares a response.
   * The response is then forwarded back to **jbase\_agent**, which sends it to **jremote**.
6. **Response Handled by OFSMessageMDB**:
   * **OFSMessageMDB** picks up the response from **jremote**.
   * It then drops the response onto the **response queue** in the J2EE application server.
7. **Response Forwarded to the T24 BrowserServlet**:
   * Finally, the **response** is forwarded from the **response queue** to the **T24BrowserServlet**, completing the request-response cycle.

**Backward Compatibility with TCServer in R09**

In **R09**, the architecture still maintains the ability to communicate with **TCServer** for backward compatibility. This ensures that older setups using **TCServer** are not disrupted and can still function with the new architecture.

The following components are involved in this process:

1. **TOCF Plugin**:
   * The **TOCF Plugin** is a **resource adapter** that handles all **TOCF(EE) listeners**, including the **BatchFileListener**.
   * This resource adapter **wraps the existing TCServer plugins**, allowing custom listeners to be used without needing to modify the code.
   * The **tcserver.xml** file inside the plugin is used to define and configure the listener interfaces.
2. **TOCF Listener MDB**:
   * The **TOCF Listener Message-Driven Bean (MDB)** is called when a message arrives at the resource adapter configured to trigger it.
   * In **R09**, the **TOCF Listener MDB** processes incoming messages, places them onto the **OFS request queue**, and waits for a response on the **OFS response queue**.
3. **R10 Update**:
   * Starting from **R10**, the **TOCF Plugin resource adapter** and **TOCF Listener MDB** are packaged inside the **tocfee.ear** file, simplifying the deployment and integration of these components.

**Network Dispatcher**

The **Network Dispatcher** is a third-party load balancing software used to distribute **HTTP requests** across multiple web servers. Here's how it enhances the system:

1. **Intelligent Load Balancing**:
   * The **Network Dispatcher** helps balance the load by distributing incoming HTTP requests based on server availability, workload, and user-defined criteria.
2. **Improves Performance and Throughput**:
   * By spreading requests across multiple servers, the **Network Dispatcher** optimizes the performance of the system and helps avoid overloading a single server.
3. **Failover Mechanism**:
   * The dispatcher can be configured with a **backup** server, ensuring that there is **no single point of failure**. If one server goes down, the backup server can take over seamlessly.

**Summary of the T24 Architecture and Components**

Now that we’ve covered the detailed request flow and additional components, let’s summarize the key elements of T24’s **multi-tier architecture**:

1. **Client Layer**:
   * **Web Browser** (Internet Explorer, Mozilla Firefox) is used as the client interface to access T24.
2. **Application Server Layer**:
   * Includes components like **Application Server Software**, **T24 Browser Servlet**, and **jremote** to handle requests from the client and forward them to the T24 Application Server.
3. **T24 Application Server Layer**:
   * Comprises **jbase\_agent**, **TAFC (Temenos Application Framework)**, and **Direct Connect Drivers** to manage the business logic and interact with external databases.
4. **Database Server Layer**:
   * T24 supports multiple database systems like **Oracle**, **DB2**, **SQL Server**, or **jBASE** to store data. **Clustering** is supported for **Oracle/DB2** for resilience.
5. **Optional Components**:
   * **Network Dispatcher**: For **load balancing** and improved performance.
   * **Message Queuing Software**: (JMS architecture) to improve **resilience** and **asynchronous communication**.
6. **Backward Compatibility**:
   * **TOCF Plugin** and **TOCF Listener MDB** ensure that legacy systems using **TCServer** are supported in the **R09** architecture.

**What You Learned in This Unit:**

1. You can now **describe** the changes to **T24 Architecture**.
2. You are able to **identify** the key components of T24’s **multi-tier architecture**:
   * **Client Layer**
   * **Application Server Layer**
   * **T24 Application Server Layer**
   * **Database Server Layer**
3. You can **analyze** optional components like:
   * **Network Dispatcher** (for load balancing)
   * **Message Queuing software** (for resilience)

The **JMS architecture** in T24 enhances reliability and scalability, and components like the **Network Dispatcher** and **TOCF Plugin** help integrate and improve the overall system’s performance.

**1. Web Browser**

A **web browser** is a software application used to access websites and web-based applications. It allows users to view and interact with content on the internet. Examples of web browsers are **Internet Explorer**, **Mozilla Firefox**, **Google Chrome**, **Safari**, etc. In the context of T24, a web browser acts as the client interface, meaning it’s the front-end that users interact with to access T24 software.

**2. Application Server Layer**

An **application server** is a platform used to host and manage applications. It provides an environment where web applications can run and exposes business logic for client applications to interact with. Unlike a simple web server, it supports complex operations such as load balancing, database connection pooling, and transaction management. In the T24 architecture, the application server processes client requests, handles business logic, and interacts with the database server.

Popular application servers are:

* **JBoss**
* **WebLogic**
* **WebSphere**

**3. T24 Browser Servlet**

The **T24 Browser Servlet** is part of the T24 application. It is responsible for handling incoming HTTP requests from the web browser, transforming them into specific internal requests that the T24 system can process. Essentially, it acts as an intermediary between the user's browser and the back-end systems.

**4. jremote (TAFC JCA)**

**jremote** is a component that helps the T24 browser communicate with the T24 application server. It's a **resource adapter** in the **Java Connector Architecture (JCA)**, a Java standard for connecting to enterprise information systems. jremote sends requests from the T24 browser to the T24 application server and receives responses.

**TAFC** refers to **Temenos Application Framework for jBASE**, which is the environment in which T24 runs, specifically the jBASE runtime system.

**5. jbase\_agent**

The **jbase\_agent** is a server-side component of the T24 system that processes requests from the client (web browser). It accepts and processes the client’s requests via the **jremote** component. The jbase\_agent listens on specific TCP/IP ports for incoming messages and interacts with the T24 business logic. This is essentially the communication bridge between the T24 application server and the T24 database server.

**6. T24 Application Server (TAFC)**

This is the main server running the business logic of T24. It’s built to process requests related to T24's operations, handle transactions, and manage the application runtime environment. The T24 application server can run on **jBASE** (as in **TAFC**) or **Java Runtime** (as in **TAFJ**).

* **TAFC** is the environment running on the jBASE runtime, which is the core platform for T24. jBASE is a high-performance, multi-dimensional database system.

**7. Direct Connect Drivers**

**Direct Connect Drivers** allow T24 to interact with external databases such as **Oracle**, **DB2**, or **SQL Server**. When T24 needs to access or manipulate data in a non-jBASE database, it uses these drivers to convert T24 requests into SQL queries that the external database can understand. Similarly, it translates responses from the database back into T24-compatible formats.

**8. Database Server Layer**

This layer stores all the data of the T24 system. T24 can work with multiple types of databases:

* **Oracle**
* **DB2**
* **SQL Server**
* **jBASE**

If T24 uses a relational database like Oracle or SQL Server, **Direct Connect Drivers** are used to bridge communication. In T24, the data is typically stored in **XML format**. **Clustering** ensures high availability by maintaining multiple copies of the database.

**9. OFS (Open Financial Services)**

**OFS** is a module within T24 that allows it to interact with external systems. It uses a **Request-Response** model to process messages. External systems can send requests to T24 using OFS, and T24 will send back a response. It ensures that external systems can interact with T24 without directly accessing the internal application logic.

**10. JMS (Java Messaging Service)**

**JMS** is a Java-based API that allows applications to send and receive messages between different components of a distributed system. It's commonly used for integrating services, ensuring **reliable message delivery** between different systems or parts of an application. In the T24 architecture, JMS is used for handling communication between components in a more **asynchronous** manner (message queues).

**OFSMessageMDB** (Message-Driven Bean) acts as a listener that listens to messages in the **JMS queue**, processes them, and forwards them to T24.

**11. TOCF Plugin**

The **TOCF Plugin** is a resource adapter in the T24 architecture. It acts as a wrapper for existing **TCServer** (Temenos Communication Server) plugins, allowing custom listeners and functionality to be added without modifying the core T24 code. The TOCF Plugin ensures that legacy systems using **TCServer** can still be integrated with the newer T24 R09 architecture.

**12. TOCF Listener MDB**

This is a **Message-Driven Bean** (MDB) that listens for messages on the **TOCF Plugin**. Once a message is received, it processes it and moves it to the **OFS request queue** for further handling by T24.

**13. Network Dispatcher**

A **Network Dispatcher** is a third-party software used to distribute incoming requests to multiple web servers. It is a **load balancing** tool that ensures requests are distributed evenly across available servers, improving the system’s performance and reliability. If one server goes down, the dispatcher can send requests to another server, ensuring high availability.

**14. Message Queuing Software**

Message queuing software (like **JMS** or **IBM MQ**) is used to manage messages in queues, ensuring that messages can be reliably delivered even if the receiver is temporarily unavailable. The messages can be processed asynchronously, meaning that the sender does not need to wait for the receiver to process the message.

**15. Enterprise Archive (EAR)**

An **EAR (Enterprise Archive)** file is a Java package that bundles multiple modules (like **EJBs**, **JARs**, and **WARs**) into a single deployable archive. In T24, it contains the required components for deployment on J2EE application servers, including the **OFSMessageMDB** and the **TOCF Plugin**.

Let me know if you'd like any more clarification on these terms!

**You said:**

TCServer?

**ChatGPT said:**

ChatGPT

**TCServer** (Temenos Communication Server) is an older component used in the T24 architecture to handle communication between the T24 system and external systems. It acts as a middleware or an interface layer that facilitates the transfer of data and messages between T24 and other external applications or systems.

Here’s a more detailed explanation of its role and functions:

**Key Functions of TCServer:**

1. **Message Handling**: TCServer was responsible for handling communication messages between T24 and external applications or other systems in the banking ecosystem. It received messages, processed them, and forwarded them to the appropriate destination (like the T24 application server or external system).
2. **Legacy Communication**: TCServer was used in older versions of T24, primarily before the introduction of newer architectures like **JMS (Java Messaging Service)** in the R09 architecture. It enabled communication using proprietary formats and protocols that were used by external systems to interact with T24.
3. **External Interface Integration**: External systems (such as third-party applications or other banking platforms) could communicate with T24 via TCServer using **Request-Response** mechanisms. TCServer would accept the incoming requests, process them, and send the corresponding responses back.
4. **Support for T24 Plugins**: TCServer supports the use of plugins. These plugins are used to extend the functionality of TCServer and to support custom integration scenarios. For instance, a plugin might handle a specific type of messaging or business logic required by an external system.

**TCServer vs. Newer Architectures (e.g., JCA/JMS)**

* **JCA/JMS (Java Connector Architecture/Java Messaging Service)**: These newer systems, introduced with T24 R09 and beyond, replace TCServer for message handling and communication. They offer **better resilience, scalability, and flexibility** compared to TCServer. JCA (used with **jremote**) and JMS allow for asynchronous communication, **reliable message delivery**, and **improved fault tolerance**.
* **Backward Compatibility**: Even after the introduction of the newer JCA/JMS architecture, T24 R09 still provides the ability to use TCServer for backward compatibility. This means existing systems that were previously integrated with TCServer could continue to interact with T24 using the old system, while new systems could leverage the newer architecture.

**Components of TCServer:**

TCServer is typically used in conjunction with the following components:

* **TOCF Plugin**: A resource adapter that wraps TCServer’s functionality, allowing it to be used with newer versions of T24.
* **TOCF Listener MDB**: An **EJB** (Enterprise JavaBean) that processes messages from external systems and sends them to T24 via TCServer.

**How TCServer Fits Into the T24 Architecture:**

* **Legacy Integration**: In previous versions of T24, TCServer played a significant role in connecting T24 to external systems.
* **Bridge to External Systems**: It acts as a bridge that receives incoming requests from external systems, processes them, and forwards the responses back to those systems. It was often used for **batch processing** and integrating with **mainframe systems**.

**Why the Move from TCServer to JCA/JMS?**

1. **Scalability**: JCA and JMS allow for better scaling, as the system can handle a large volume of requests asynchronously and distribute the load more effectively.
2. **Resilience**: The newer systems offer more robust error handling, ensuring that messages are not lost even if one part of the system fails.
3. **Standardization**: JCA and JMS are industry-standard technologies that are more flexible and compatible with modern infrastructure.