Introduction to Component Architectures and Java EE

Authors: Simon Pickin
Natividad Martinez Madrid

Address: Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid
Spain

Version: 1.1

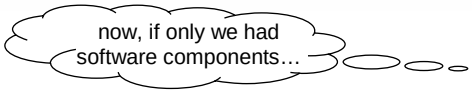
Communication Software 2010-2011

Contents

Communication Software 2010-2011

© The Authors

Origin of Software Component Ideas



- NATO conference on S.E.
  - Garmisch, Germany, 1968
  - to counter the “software crisis”
  - see **`http://homepages.cs.ncl.ac.uk/brian.randell/NATO/`**
- Doug McIlroy’s address at this conference:
  - title: *Mass produced software components*
  - topic: software should be built from prefabricated components
  - for complete text, see: **`http://cm.bell-labs.com/cm/cs/who/doug/components.txt`**

Definition of Component (1/5)

- A unit of composition with contractually specified interfaces and fully explicit context dependencies. A software component can be deployed independently and is subject to third-party composition.

  Clemens Szyperski
  Component Software: Beyond Object-Oriented Programming

Definition of Component (2/5)

- Software components enable practical reuse of software parts and amortization of investments over multiple applications. There are other units of reuse, such as source code libraries, design, or architectures. Therefore, to be specific, software components are binary units of independent production, acquisition, and deployment that interact to form a functioning system.

Clemens Szyperski
Component Software : Beyond Object-Oriented Programming)

Definition of Component (3/5)

- … represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A component defines its behavior in terms of provided and required interfaces. As such, a component serves as a type whose conformance is defined by these provided and required interfaces (encompassing both their static as well as dynamic semantics)… A component is modeled throughout the development lifecycle and successively refined into deployment and run-time.

OMG
UML superstructure version 2

Definition of Component (4/5)

- What is a component? A software module that:
  - is oriented towards the development of applications by assembly of existing modules
  - that facilitates the division of work (clear responsibilities)
  - is a unit of reuse: can be chosen off-the-shelf, ready for use (COTS: *Components "Off-The-Shelf"*)
  - is a unit of deployment: can be independently compiled and deployed

Definition of Component (5/5)

- We may also require of a component that:
  - it be part of a distributed application
  - it can be used in a flexible manner ⇒ multiple interfaces
  - it can communicate in a flexible manner, e.g.:
    - synchronous communication by method invocation
    - asynchronous communication via event channels

Component Models

- What does a component model comprise?
  - a notion of individual component
  - a definition of how to assemble components
    - by connecting compatible interfaces
  - a definition of a component environment
    - i.e. an execution and deployment environment for the components
- What does a component model provide?
  - design and development by assembly; reuse
  - a clear view of the application architecture
  - separation of functional and non-functional aspects
- The above requirements are not fulfilled by:
  - object-based models such as RMI,...
  - service-based models such as CORBA 2, Jini,...

Distributed Component Environments (1/2)

- What is a distributed component environment?
  - an environment conceived for the deployment and execution of component-based distributed applications

Distributed Component Environments (2/2)

- What does a distributed component environment involve?
  - the separation of functional and non-functional aspects
  - the *implicit* management and support by the execution env. (via standard profiles) of non-functional aspects such as:
    - security (authentication, authorization,...)
    - transactions (declarative definition o via API)
    - concurrency control
    - persistence (environment-managed or via API)
    - life-cycle management
    - component naming, trading and search services
    - activation & deactivation/"passivation"
    - communication protocols
    - component administration
  - support for deployment

Component Based Development (CBD)

- Currently, various claims on CBD, e.g.:
  - Enterprise JavaBeans and Java EE component model
  - .NET component framework
  - CORBA 3 and CORBA Component Model (CCM)
- Currently, component definitions not quite satisfied:
  - Szyperski: "with… fully explicit context dependencies"
  - UML 2.0: "a component defines its behavior in terms of provided and required interfaces"
- Component architecture / component framework
  - synonyms of term component model (usually)

Components vs. Objects

- Commonalities:
  - modularity (low external coupling, high internal cohesion)
  - encapsulation/information hiding, abstraction,…
- Components
  - have greater granularity than objects
  - are more associated to the application than objects
    - not viewed as corresponding to real-world elements
  - have explicit context dependencies
    - current implementations : not all such dependencies are explicit
  - interact via well-defined interaction protocols
  - constitute units of deployment: can be independently deployed
  - constitute units of re-use (but don´t class libraries?)
    in any case, they should lead to greatly-increased re-use

Enterprise Application Requirements (1/2)

- ***Storage of, and access to, data (back-end integration):*** employment of database systems (DBMS), database connectivity, representation of data in the database
- ***Data mapping and persistence*****:** representation of data in programs (classes) and correspondence (mapping) to its representation in the data base, update of the database by the program after changes
- ***Data integrity*****:** control of concurrent access to data, transaction monitors
- ***User interaction*****:** authentication, access control, coordination of concurrent access
- ***Access to shared data*****:** isolation of different accesses, data caching

Enterprise Application Requirements (2/2)

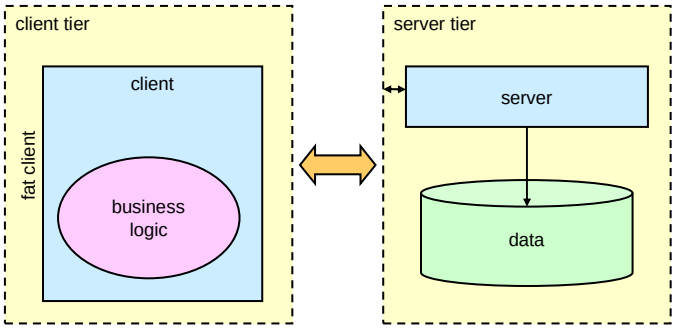
- ***Performance*:** good response time, efficient interaction between different components
- ***Scalability*:** possibility of incorporating new servers, load distribution
- ***Availability*:** security on application crashes (ideally 24 x 7 availability), fault tolerant systems, server and data clustering
- ***Software design*:** mantainability, portability and interoperability
  → modularity, design in levels, reduced external dependence (e.g., on the data base)
- ***Platform-independence***?

Contents

Classic Client/Server Architectures

- Application divided functionally and physically (subsystems) into two tiers: client and server
  - business data – in a data base, or in general an *Enterprise Information System* (EIS) – resides on the server
- Two main types according to where execution logic, preparation and presentation of information, interaction with the user etc. resides
  - **fat client**: main part of application executes on client
  - **thin client**: main part of application executes on server
- Client and server are weakly coupled and communicate only via messages
- Role of server is passive: communication initiated by client (with a service request); server responds

Classic Client/Server: Fat Client



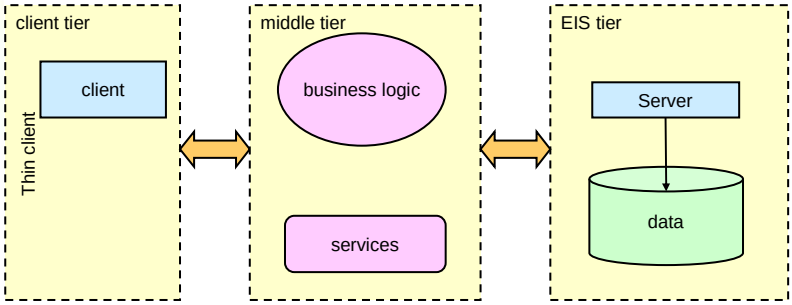
Advantages of thin clients

- Less infrastructure needed at the client
  - reduce costs since typically there are many clients & few servers
- Easier administration
  - that is, configuration, maintenance, deployment,…
  - given that there are more clients than servers
- Less network traffic
  - since client is offered a more abstract level of service
- Centralised resource management
  - helps to ensure data integrity
  - more secure
  - better fault detection
  - more control over transactions
  - …
- More easily evolved, e.g. change of DBMS

Multi-Tiered Architectures (1/2)

- In multi-tier architectures (3-tier / n-tier), additional software tiers are added each dealing with specific tasks
  - clients are thin clients
- The intermediate tiers extend the responsibility of the server side
  - though may be situated in independent nodes or systems
- Each tier communicates
  - only with the contiguous tiers
  - via clearly defined interfaces

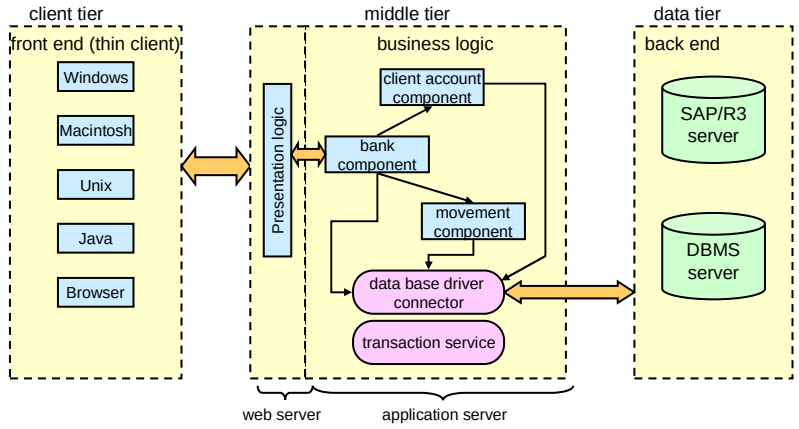
Multi-Tiered Architectures (2/2)



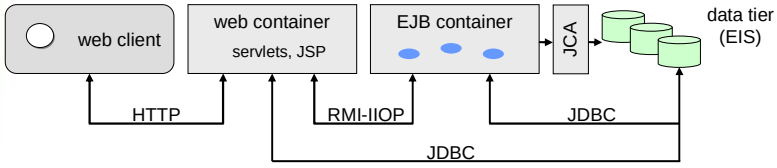
Advantages of multi-tier architectures

- All the advantages of thin clients
- More flexibility and scalability
- Tiers can be updated / replaced independently
  - on changes in requirements or in technology
- Greater control of the server load
  - avoid overloading the server
  - load-balancing between servers
  - obtain lower response time (in general)
- Debugging easier
  - due to increased modularity

Example: Multi-Tiered Architectures



Java EE Web Application Architecture



Application Server (and Container)

- Support system for server components
  - development environment for components
  - the server components use the services of the application server
- Infrastructure tasks:
  - component instantiation
  - communication
  - synchronization of concurrent access
  - preparation of a secure environment
  - availability
  - transaction security

Client Tier

- The part of the application that executes on the client's computer
- Usually implements the following functions:
  - present the information coming from the server
  - collect input data
- Usually is a web browser; may include:
  - applets to show graphic information
  - Javascript to pre-process input
  - plug-ins (such as Flash)

  or may be a Java application

Middle Tier

- Application server: principal part of the application
  - application and business logic
  - preparation of the information for the user
- Middleware
  - usually includes specialised software for implementing certain tasks (standard enterprise services) :
    - monitors, naming systems, message queuing systems, etc.
    - e.g. CORBA and CORBA services

Middle Tier: Presentation logic

- Receives the requests from the clients
  - extracts the client data
  - extracts additional information (request headers)
- Performs some pre-processing of the request
  - decides what business-level services are necessary
  - calls the required business-level services
- Prepares the client responses
  - response headers
  - contents of the response (typically HTML)

Middle Tier: Business logic

- Implements the business logic
  - i.e. the functionality itself
- Application server
  - may also integrate the presentation part of the application
- Application server contains:
  - a container where the business components “live”
    - session components (represent processes)
    - entity components (represent data)
  - the other middleware services
    - transaction-based data processing
    - secure access
    - monitors
    - naming system, etc.

Data Tier

- Data bases or “Enterprise Information Systems”
- Responsible for the administration of, rapid access to, and persistence of data
- Accessed by the business tier
  - data mapping, from the business-tier data representation to the EIS representation, required
- Note: very simple web applications may have no business tier
  - presentation tier includes the application logic
  - presentation tier communicates directly with data tier

Contents

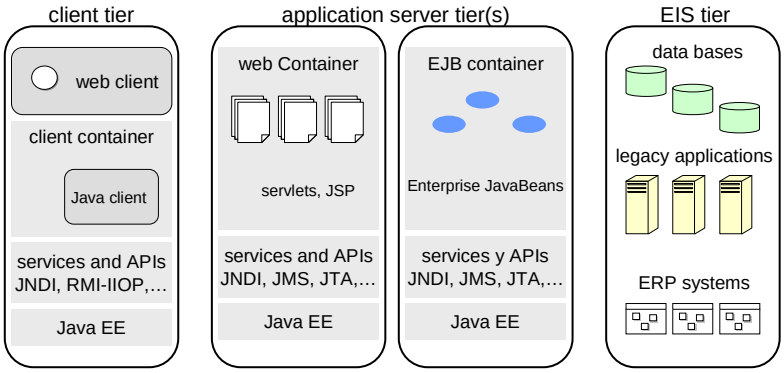
The Java Enterprise Edition Platform

- Collection of specifications and programming directives to facilitate the development of internet-enabled multi-tier, distributed server applications.

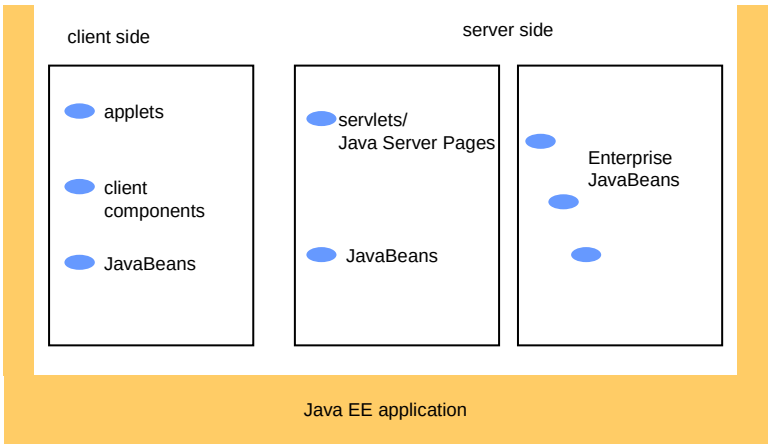
Some history…

- 1996: Java Development Kit (JDK) 1.02: an ordered collection of class libraries and packages
- 1999: JDK 1.2 → Java 2 Platform: in addition to the JDK, optional packages for messaging, dynamic web page generation and e-mail programs in Java. Divided into 3 editions:
  - *Java Standard Edition* (Java SE): contains the current SDK and the standard APIs; aimed at desktop applications and applets
  - *Java Enterprise Edition* (Java EE): based on Java SE, extends the server side; v1.3 (I2001), v1.4 (2003), Java EE5 (2006), Java EE6 (dec. 2009)
  - *Java Micro Edition* (Java ME): aimed at mobile and embedded systems: mobile telephones, pagers, palmtops, etc.

Java EE Application Architecture

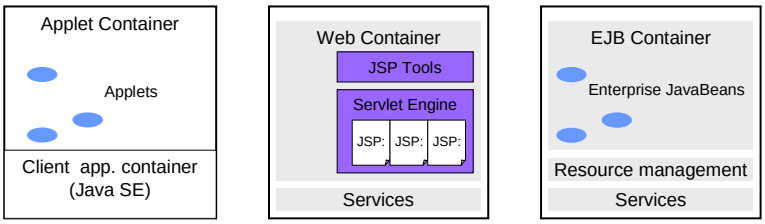


Types of components in Java EE



Container

- Offers an execution environment for the application components
- Provides a uniform view of the services required by the components
  - as specified in the component descriptors (spec. of cmpnt. dependencies)
- May also provide deployment tools
  - for installation & configuration of components (including at execution time)
- Principal tasks, server side containers:
  - resource and lifecycle management



Servlets and JavaServer Pages

- Servlets and JavaServer Pages (JSP) are presentation-level components
- They perform dynamic generation of web pages
- Servlets:
  - Java code: easier to control the flow of actions
- JSP:
  - mark-up language based on tags: easier to represent information
- Equivalent: a JSP translates to a servlet

Enterprise JavaBeans

- Enterprise JavaBeans (EJB) is a complete service-component architecture specification (component model)
- Objectives of the EJB component architecture:
  - facilitate the development of applications, concentrating on the business logic: development, application and execution-time aspects
  - achieve independence of the component provider through the specification of interfaces
  - achieve platform-independence thanks to the : “Write Once Run Anywhere” (WORA) principle and its implementation in Java
  - ensure compatibility with existing Java APIs, with third-party server systems and with CORBA & Web Services protocols

What are Enterprise JavaBeans?

- See the definition of a component model (slide 10)
- Notion of individual component. EJBs are:
  - components used as part of distributed enterprise applications
  - encapsulated parts of the business logic of an application
- Definition of how to assemble components. EJBs:
  - communicate with resource managers and other EJBs
  - are accessed by different types of clients: EJBs, servlets, application clients, etc.
- An execution environment (the EJB container).
  - EJB container provides access to services for security, transactions, deployment, concurrency control, life-cycle management,…
  - an application may have one or several EJB containers each containing one or several EJBs

Types of EJBs

- Entity Beans (now superceded by JPA entities):
  - model business concepts such as persistent objects associated to data, e.g.bank account, product, order…
- Session Beans:
  - represent processes executed in response to a request from the client, e.g. bank transactions, calculations, ordering…
- Message-Driven Beans:
  - represent processes executed as a response to the reception of a message

Java EE Services (1/2)

- **Naming service**: access to components/resources via logical names
  - portability and maintainability
  - *Java Naming and Directory Interface (JNDI)*
- **Transaction service**: execution of a series of steps in an atomic and isolated manner
  - *declarative transaction demarcation*: define transaction boundaries via deployment descriptors
  - *programmatic transaction demarcation*: possibility of finer transaction control via an API
  - *Java Transaction Service (JTS)*
- **Security service**: security directives for protected resources
  - two-step access control: authentication y authorisation
  - declarative or programmed implementation
  - *Java Authentication & Authorization Service (JAAS)*

Java EE Services (2/2)

- **Persistence**: persistent storage of objects and object states, normally implemented using relational data bases
  - declarative: CMP, container-managed persistence (EJB2 only), JPA, Java Persistence API (EJB3 and Java SE)
  - programmatic: Via JDBC API
- **Communication**: different communication techniques, provided by the container or application service provider
  - web communications : TCP/IP, UDP/IP, HTTP y HTTPS (with SSL/TLS)
  - distributed object processing: RMI (Remote Method Invocation)
    - Java Remote Method Protocol (JRMP)
    - CORBA-IIOP for interoperability between Java EE and CORBA
    - JAX-WS/JAX-RPC for interoperability between Java EE and Web Services
- **Configuration / administration services**: packaging, installation and flexible configuration of components and administration of applications
  - description of the characteristics of servers, containers, applications, components and services using XML schemas