SOFTWARE ENGINEERING
(Week-9)

USAMA MUSHARAF

MS-CS (Software Engineering)

LECTURER (Department of Computer Science)

FAST-NUCES PESHAWAR

AGENDA OF WEEK # 9

- Architectural Styles (Cont..)
- Data Centered/shared Software Architecture
  - Repository
  - Blackboard
- Component based Software Architecture

CATEGORIES OF ARCHITECTURAL STYLES

- **Hierarchical Software Architecture**
  - Layered
- **Data Flow Software Architecture**
  - Pipe n Filter
  - Batch Sequential
- **Distributed Software Architecture**
  - Client Server
  - Peer to Peer
  - REST
  - SOA
  - Microservices
- **Event Based Software Architecture**
- **Data Centered Software Architecture**
  - Black board
  - Shared Repository
- **Component-Based Software Architecture**

DATA CENTERED / SHARED DATA SOFTWARE ARCHITECTURE

SHARED DATA SOFTWARE ARCHITECTURE

- Data-centered software architecture is characterized by a centralized data store that is shared by all surrounding software components.
- The software system is decomposed into two major partitions: data store and independent software component or agents.

SHARED DATA SOFTWARE ARCHITECTURE

- In pure data-centered software architecture, the software components don't communicate with each other directly; instead, all the communication is conducted via the data store.

- The shared data module provides all mechanisms for software components to access it, such as insertion, deletion, update, and retrieval.

SHARED DATA:

- Blackboard style
- Repository style

REPOSITORY ARCHITECTURE

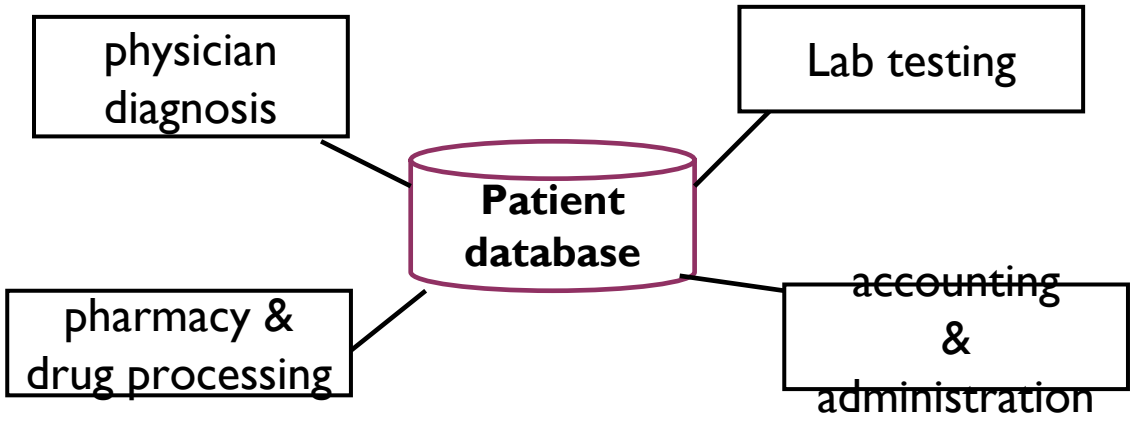
REPOSITORY ARCHITECTURE

- The repository architecture style is a data-centered architecture that supports user interaction for data processing.

- The software component agents of the data store control the computation and flow of logic of the system.

REPOSITORY ARCHITECTURE

- All data in a system is managed in a central repository that is accessible to all system components.
- Components do not interact directly, only through the repository.



REPOSITORY ARCHITECTURE

- Organizing tools around a repository is an efficient way to share large amounts of data.
  - There is no need to transmit data explicitly from one component to another.
- Although it is possible to distribute a logically centralized repository, there may be problems with data redundancy and inconsistency.
  - In practice, it may be difficult to distribute the repository over a number of machines.

EXAMPLE: INTEGRATED DEVELOPMENT ENVIRONMENT

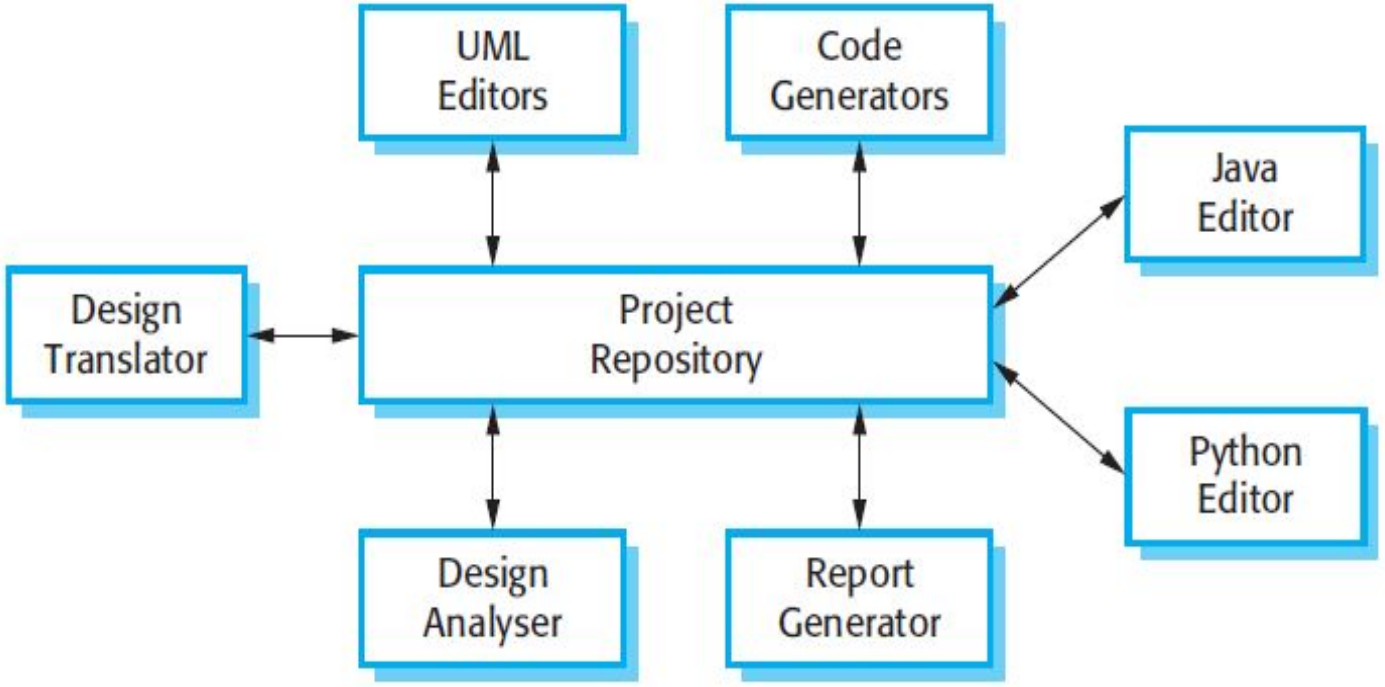


Figure 6.9 A repository architecture for an IDE

APPLICABLE DOMAINS OF REPOSITORY ARCHITECTURE:

- Suitable for large, complex information systems where many software component clients need to access them in different ways
- Requires data transactions to drive the control flow of computation

ADVANTAGES

- Components can be independent—they do not need to know of the existence of other components.
- Changes made by one component can be propagated to all components.
  - All data can be managed consistently (e.g., backups done at the same time) as it is all in one place.

DISADVANTAGES

- The repository is a single point of failure so problems in the repository affect the whole system.
- May be inefficiencies in organizing all communication through the repository.
- Distributing the repository across several computers may be difficult.

BLACKBOARD ARCHITECTURE

BLACKBOARD ARCHITECTURE

- The blackboard architecture was developed for speech recognition applications in the 1970s.
- Other applications for this architecture are image pattern recognition and weather broadcast systems.

BLACKBOARD ARCHITECTURE

- The word blackboard comes from classroom teaching and learning.
- Teachers and students can share data in solving classroom problems via a blackboard.
- Students and teachers play the role of agents to contribute to the problem solving.
- They can all work in parallel, and independently, trying to find the best solution.

BLACKBOARD ARCHITECTURE

The entire system is decomposed into two major partitions.

- One partition, called the blackboard, is used to store data.
- while the other partition, called knowledge sources, stores domain specific knowledge.
- There also may be a third partition, called the controller, that is used to initiate the blackboard and knowledge sources and that takes a main role and overall supervision control.

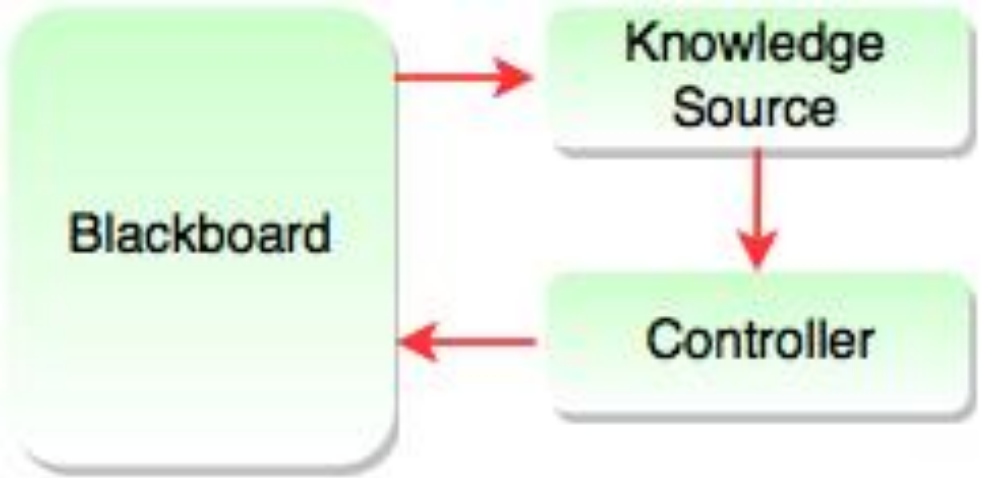
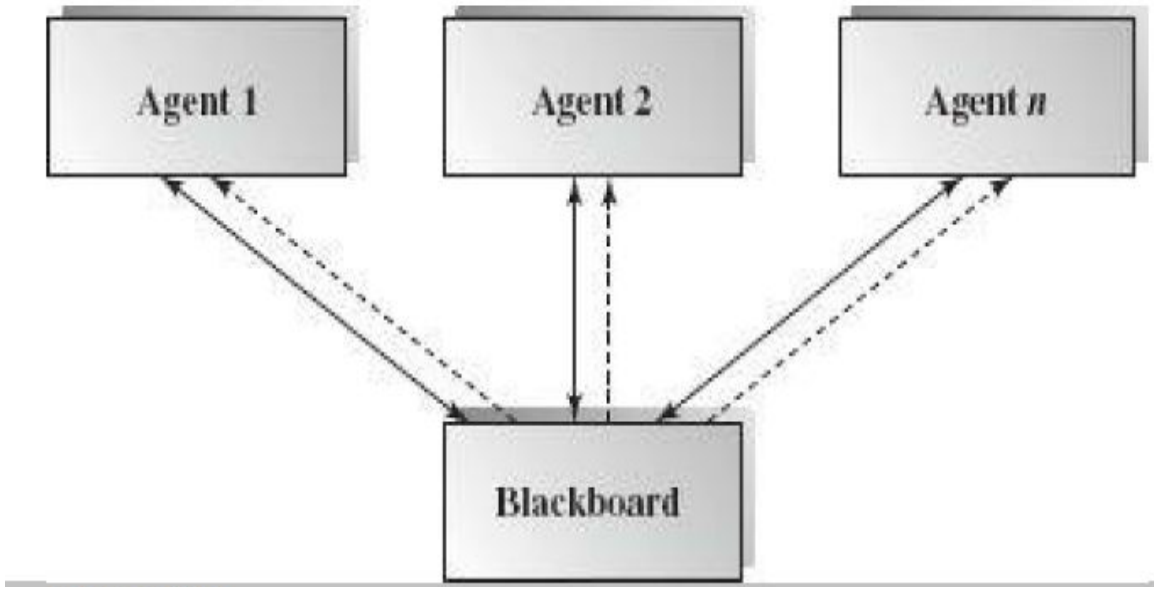


Fig. Blackboard Architectural Style

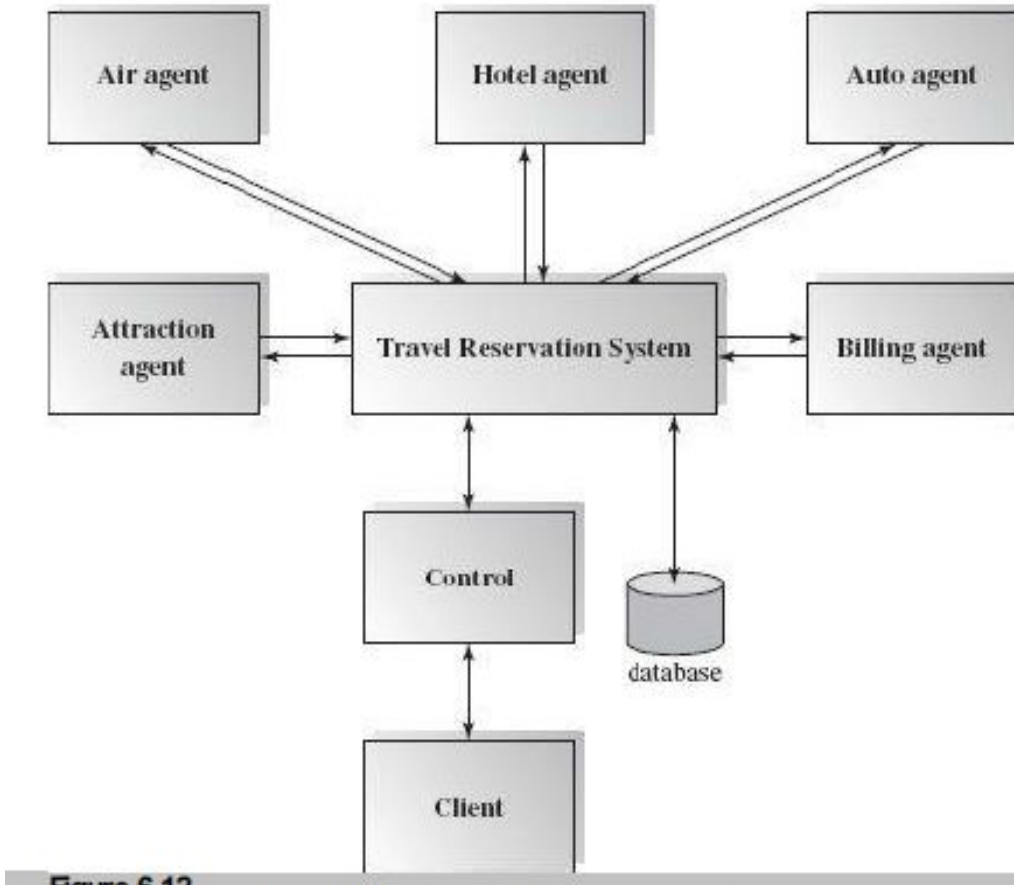
BLACKBOARD ARCHITECTURE - CONNECTIONS

- Data changes in the blackboard trigger one or more matched knowledge source to continue processing.
- This connection can be implemented in publish/subscribe mode.



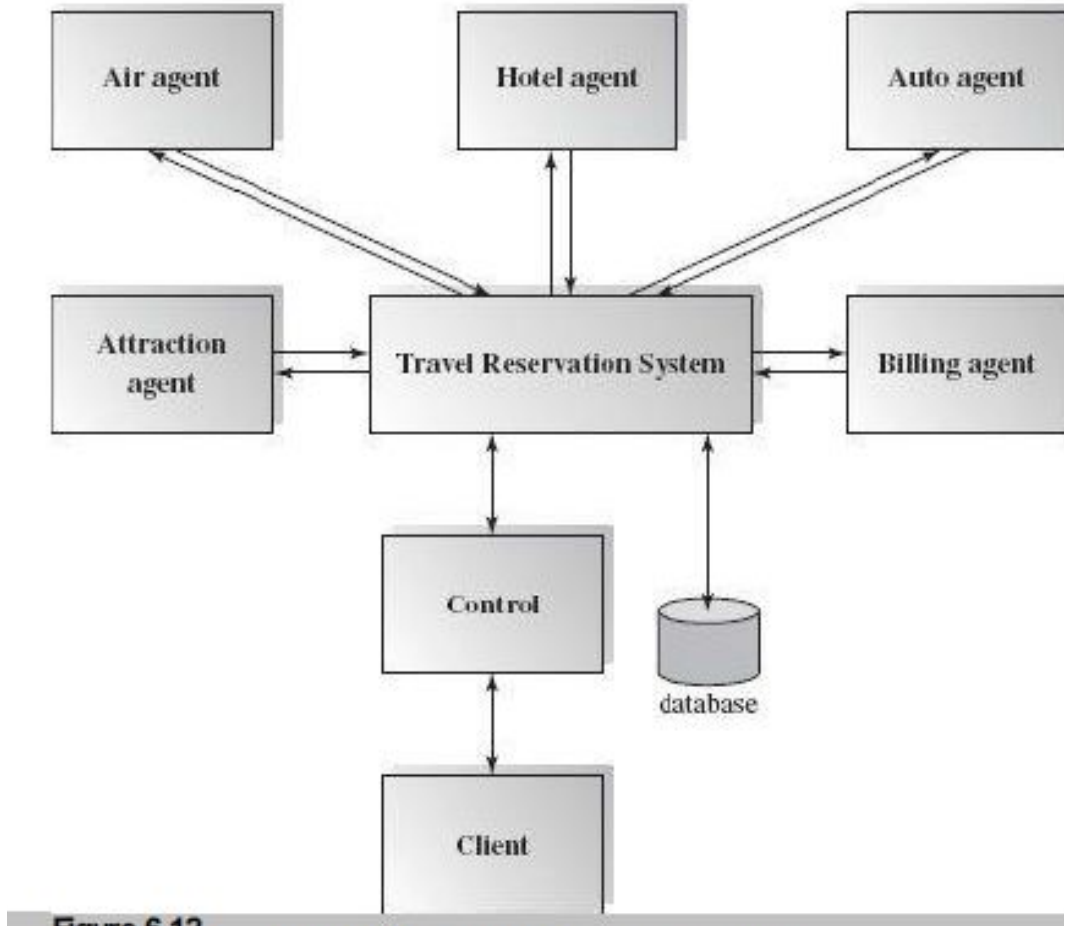
EXAMPLE - TRAVEL CONSULTING SYSTEM

There may be many air travel agencies, hotel reservation systems, car rental companies, or attraction reservation systems to subscribe to or register with through this travel planning system.



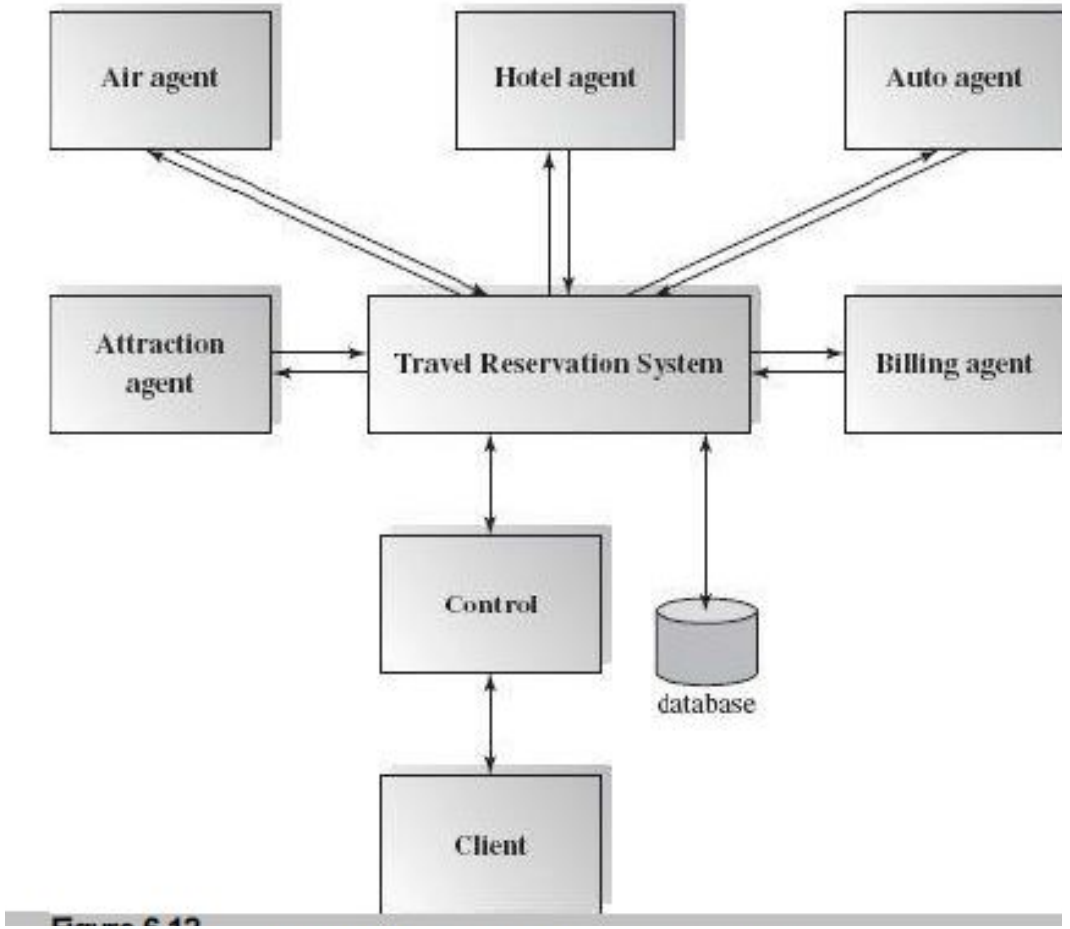
EXAMPLE - TRAVEL CONSULTING SYSTEM

- Once the system receives a client request, it publishes the request to all related agents and composes plan options for clients to choose from.



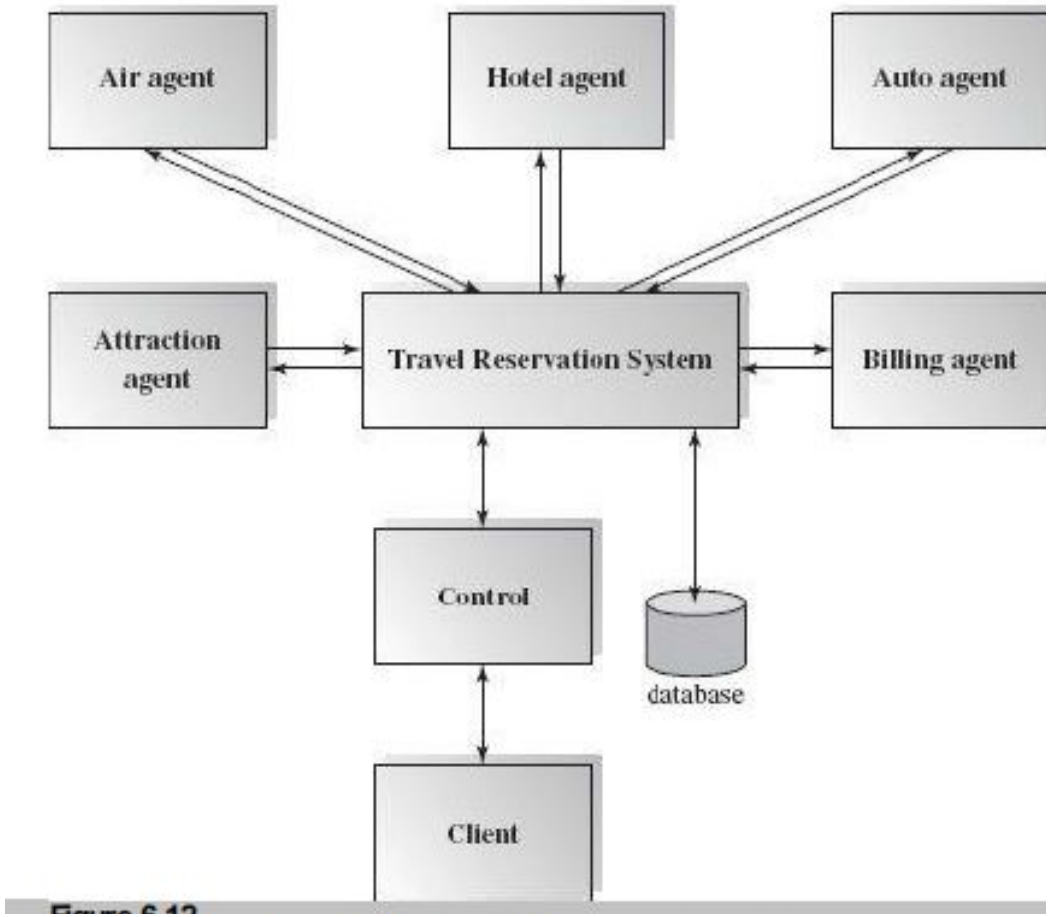
EXAMPLE - TRAVEL CONSULTING SYSTEM

- The system also stores all necessary data in the database.
- After the system receives a confirmation from the client, it invokes the financial billing system to verify credit background and to issue invoices.



EXAMPLE - TRAVEL CONSULTING SYSTEM

- The data in the data store plays an active role in this system.
- It does not require much user interaction after the system receives client requests since the request data will direct the computation and activate all related knowledge sources to solve the problem.



APPLICABLE DOMAIN:

- Suitable for solving complex problems such as artificial intelligence (AI) problems where no preset solutions exist.

BENEFITS:

- Scalability: easy to add or update knowledge source.
- Concurrency: all knowledge sources can work in parallel since they are independent of each other.
- Reusability of knowledge source agents.

COMPONENT BASED SOFTWARE ARCHITECTURE

COMPONENT-BASED SOFTWARE ENGINEERING

- CBSE is an approach to software development that relies on reuse
- CBSE emerged from the failure of object-oriented development to support reuse effectively
- Objects (classes) are too specific and too detailed to support design for reuse work

COMPONENT

- A software component is an independently deployable implementation of some functionality, to be reused as it is in a broad spectrum of applications.

- For a software component what it provides and requires should be clearly stated so that it can be used without any ambiguity.

COMPONENT BASED SOFTWARE ARCHITECTURE

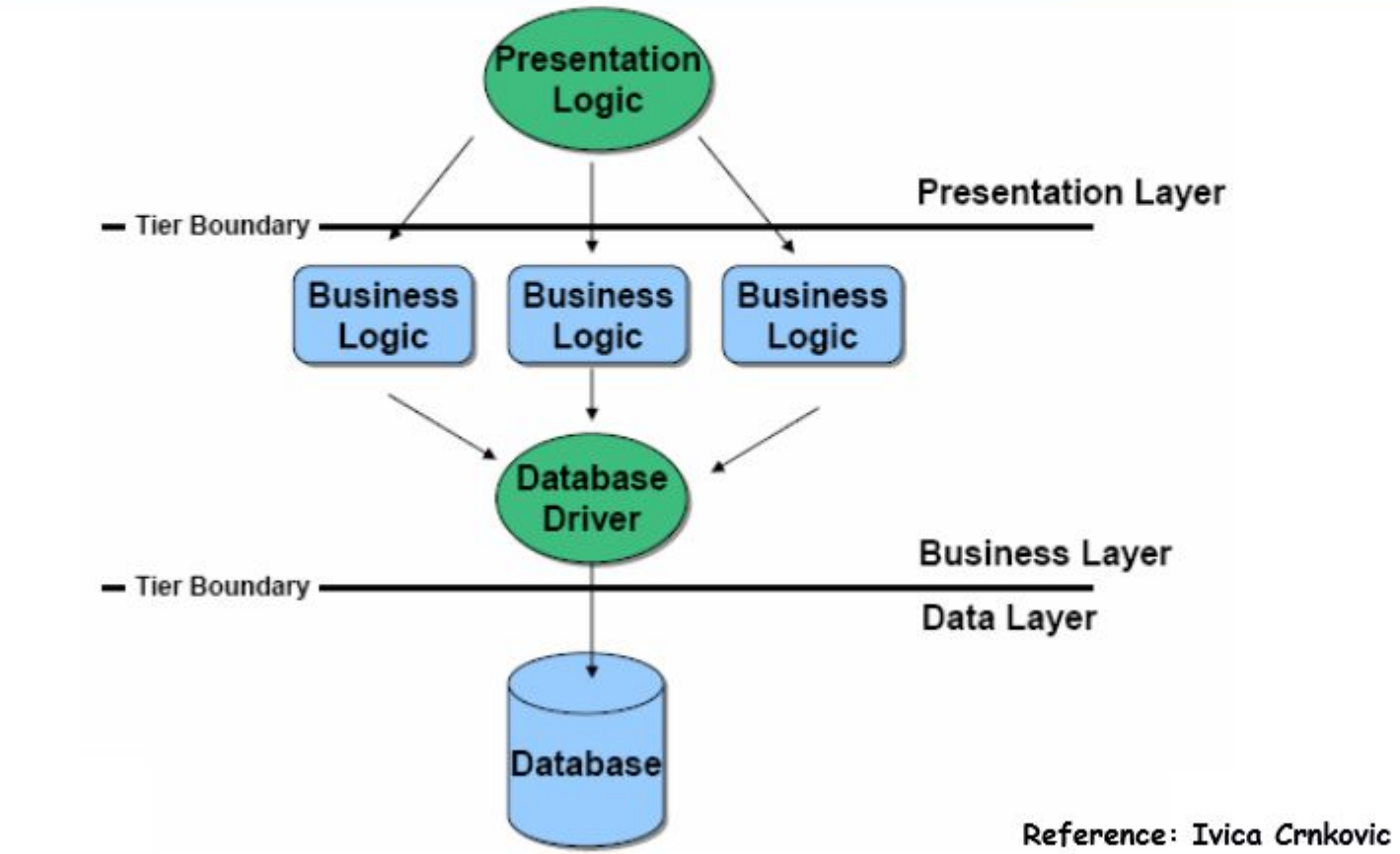
- The main motivation behind component-based design is component reusability.

- Designs can make use of existing reusable commercial off-the-shelf (COTS) components or ones developed in-house, and they may produce reusable components for future reuse.

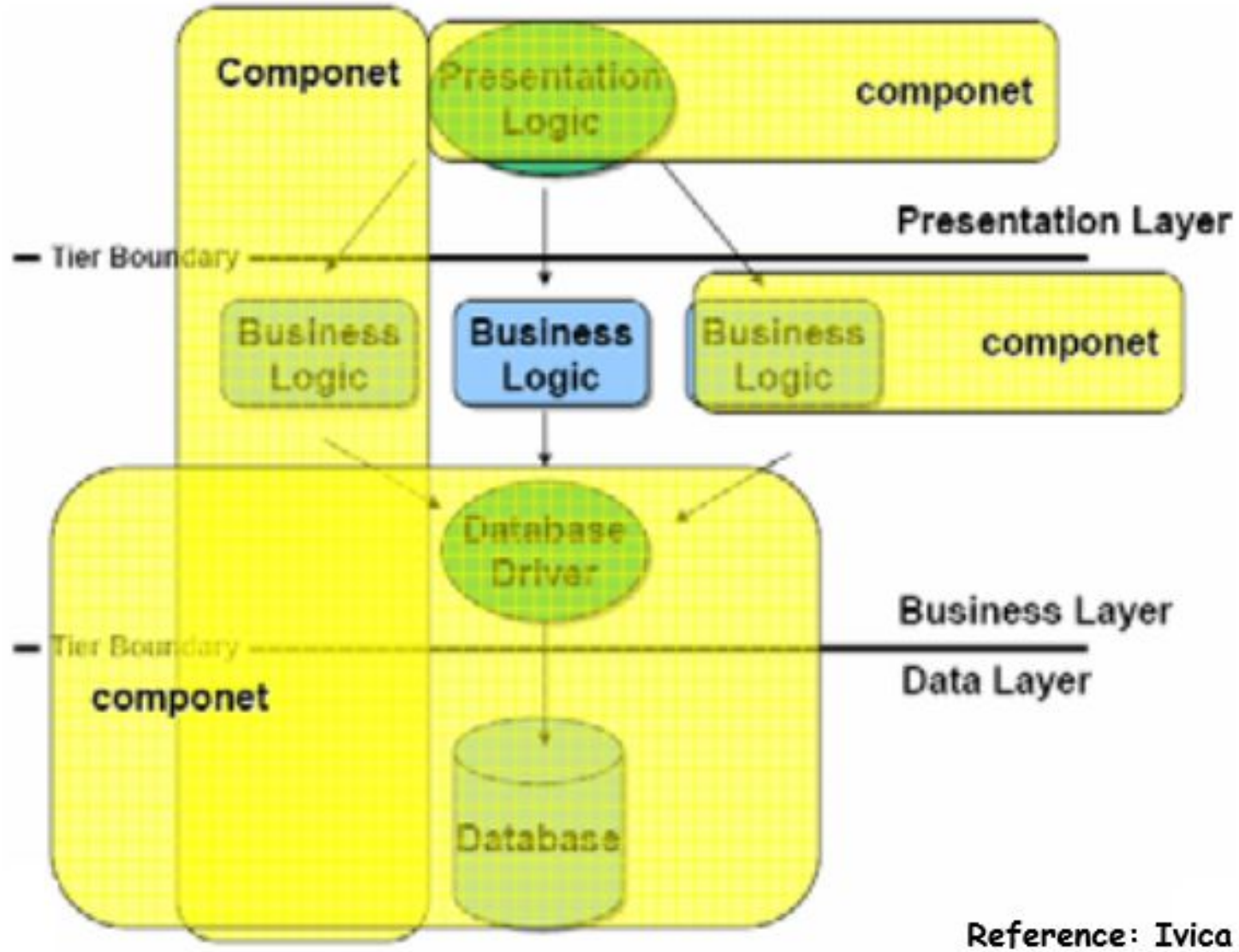
CATEGORIES OF COMPONENTS

- Commercial of-the-shelf components- complete application libraries readily available in the market.
- Qualified components—assessed by software engineers to ensure that not only functionality, but also performance, reliability, usability, and other quality factors conform to the requirements of the system/product to be built.
- Adapted components—adapted to modify (wrapping) unwanted or undesired characteristics.

N TIER ARCHITECTURE



Reference: Ivica Crnkovic



Reference: Ivica Crnkovic

CONNECTORS

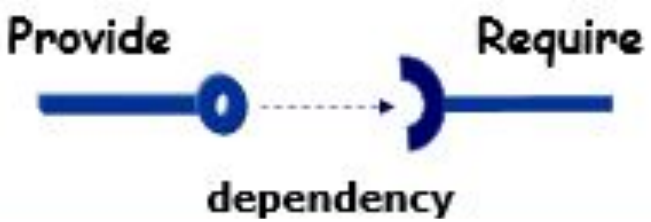
Connectors connect components, specifying and ruling their interaction.

Component interaction can take the form of

- method invocations,
- asynchronous invocations such as
  - event listener and registrations,
  - broadcasting,

Requires interface
Defines the services that the component uses from the environment

Provides interface
Defines the services that are provided by the component to other components



COMPONENT DIAGRAM

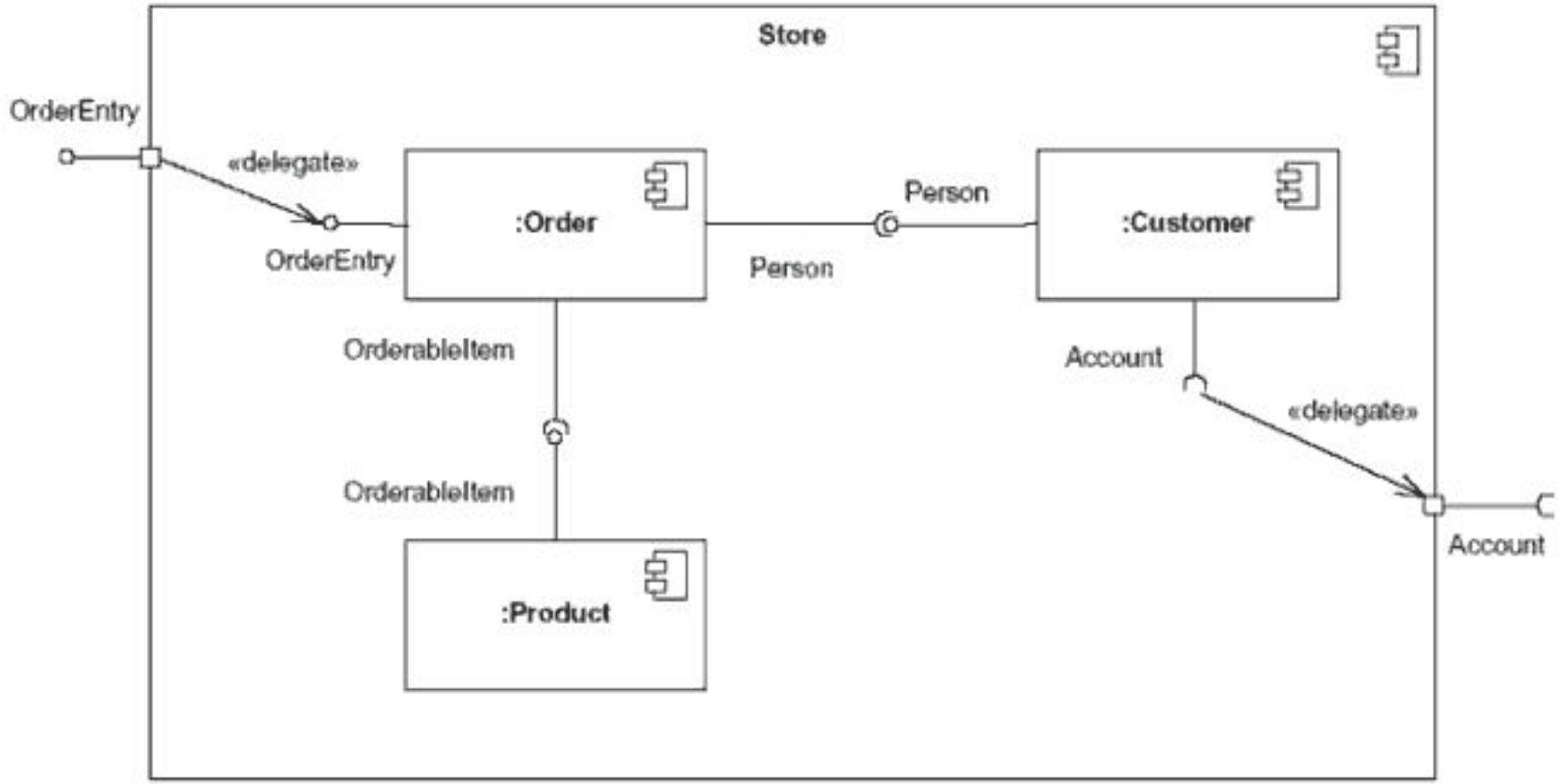
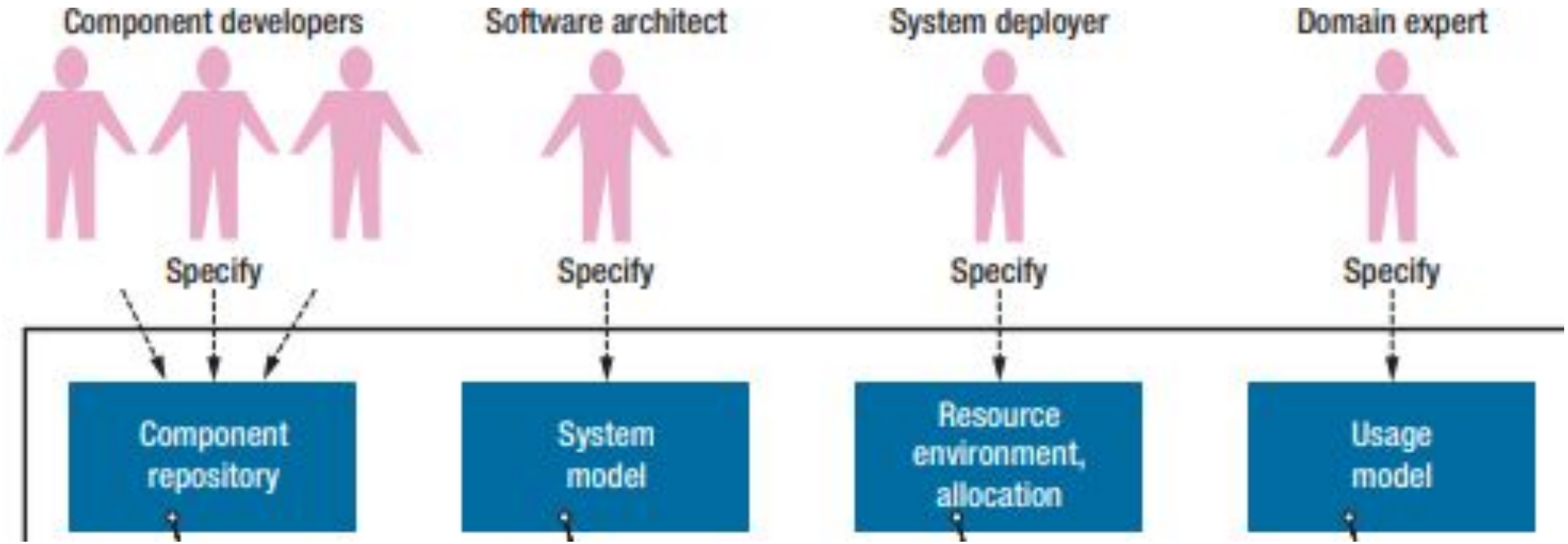
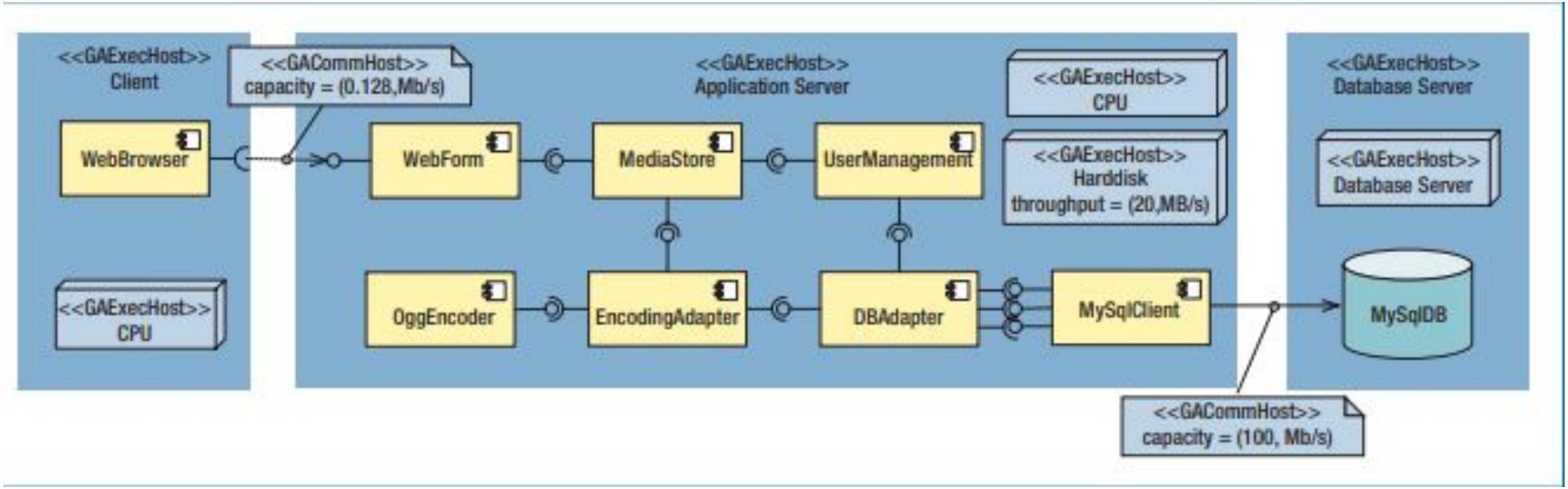


Figure: This component's inner structure is composed of other components


Component developers
Software architect
System deployer
Domain expert
Specify
Specify
Specify
Specify
Component repository
System model
Resource environment, allocation
Usage model


<<GAExecHost>>
Client
<<GACommHost>>
capacity = (0.128,Mb/s)
WebBrowser
<<GAExecHost>>
CPU
<<GAExecHost>>
Application Server
WebForm
MediaStore
UserManagement
OggEncoder
EncodingAdapter
DBAdapter
MySqlClient
<<GAExecHost>>
CPU
<<GAExecHost>>
Harddisk
throughput = (20,MB/s)
<<GAExecHost>>
Database Server
<<GAExecHost>>
Database Server
MySqlDB
<<GACommHost>>
capacity = (100, Mb/s)

COMPONENT BASED DEVELOPMENT

COMPONENT BASED DEVELOPMENT

This process basically consists of three main stages namely

- qualification,
- adaptation and
- composition

COMPONENT QUALIFICATION

Component qualification ensures that a candidate component

- will perform the function required,
- will properly fit into the architectural style specified for the system, and
- will exhibit the quality characteristics (e.g., performance, reliability, usability) required for the application.

COMPONENT ADAPTION

- Most of the times even after the component has been qualified for use in the architecture it exhibits some conflicts.

- To soothe these conflicts certain techniques such as component wrapping is used.

WRAPPING

- White-Box wrapping – this wrapping involves code level modifications in the components to avoid conflicts.

This is not widely used because the COTS products are not provided with the source code.

WRAPPING

- Grey-Box wrapping- applied when the component library provides a component extension language or API that enables conflicts to be removed.

- Black-box wrapping - introduction of pre- and post-processing at the component interface to remove or mask conflicts.

COMPONENT COMPOSITION

- Architectural style depends the connection between various components and their relationships.

- The design of the software system should be In such a way that most of the components are replaceable or can be reused in other systems.

BENEFITS

- Ideally the components for reuse would be verified and defect-free.
- As the component is reused in many software systems any defect if there would be detected and corrected. So after a couple of reuses the component will have no defects.

BENEFITS

- Productivity is increased as every time the code need not be re-written from the scratch.
- The time taken to design and write the code also decreases with the use of software components.

LIMITATIONS

- It can be difficult to find suitable available components to reuse.
- Adaptation of components is an issue.

HAVE A GOO DAY!