



HY 351: Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων CS 351: Information Systems Analysis and Design

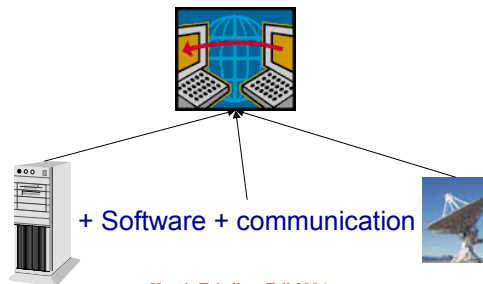
HY351:
Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων
Information Systems Analysis and Design



Σχεδίαση Φυσικής Αρχιτεκτονικής (Physical Architecture Design)

Γιάννης Τζιτζίκας

Διάλεξη : 17
Ημερομηνία : 24-1-2007
Θέμα :



Διάρθρωση

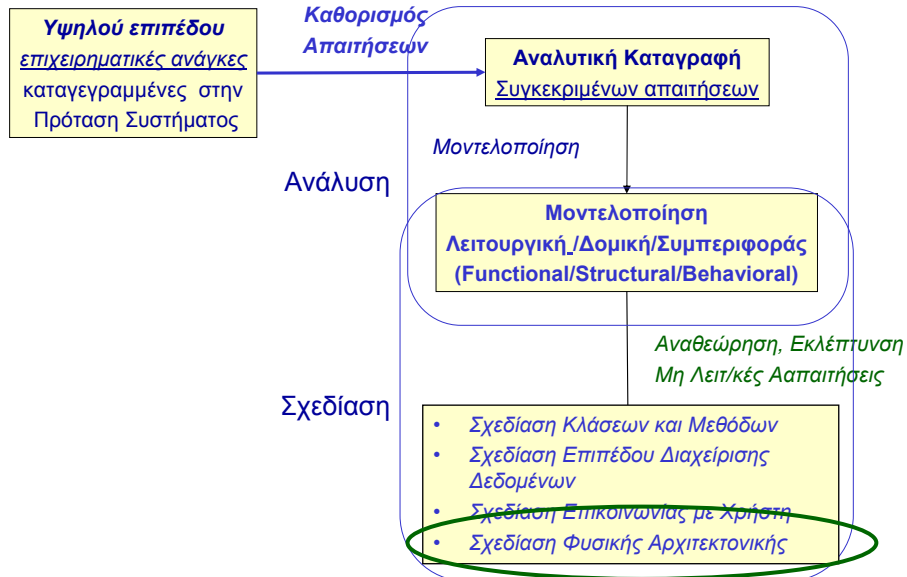
- Τι είναι η Σχεδίαση Φυσικής Αρχιτεκτονικής
- Οι 4 βασικές λειτουργίες ενός Πληροφοριακού Συστήματος
- Διαστρωματωμένες Αρχιτεκτονικές Λογισμικού
 - Layered Software Architectures
- Αρχιτεκτονικές Λογισμικού
 - Client-server, N-tier architectures, Virtual machine
 - Service-oriented computing, P2P
- Πρωτόκολλα Επικοινωνίας
- Το σχεδιαστικό μοτίβο MVC

2^ο Μέρος (Επόμενο μάθημα):

- Σχετικά Διαγράμματα της UML
 - Component and Deployment Diagrams



Από τα Μοντέλα Ανάλυσης στα Μοντέλα Σχεδίασης



Τι είναι η Σχεδίαση Φυσικής Αρχιτεκτονικής (ή της αρχιτεκτονικής του συστήματος);
System (or Physical) Architecture Design ?

System Architecture Design comprises plans for

- the **hardware**,
- the **software**,
- the **communications**

for the new application.



The 4 primary software components of a system

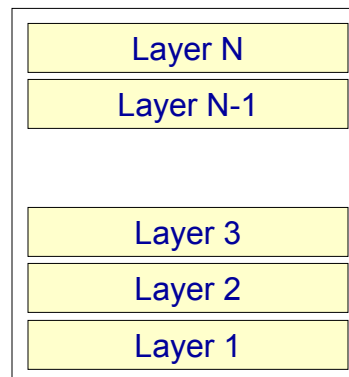
All software systems could be divided into 4 basic functions

- *Data storage*
- *Data access logic*
- *Application logic*
- *Presentation logic*



Διαστρωματωμένα Συστήματα Layered Systems

- The functionality of the application is partitioned to a set of layers
- Each layer uses the services of the lower layers and offers services to the upper layers
- **Advantages**
 - Abstraction during design
 - Allow reuse
 - Can define standard layer interfaces
- **Disadvantages**
 - Sometimes it is difficult to identify with clarity the layers.
 - Sometimes this architecture is not very efficient (redundant)

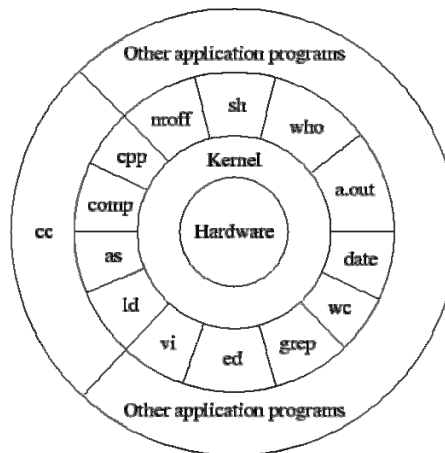
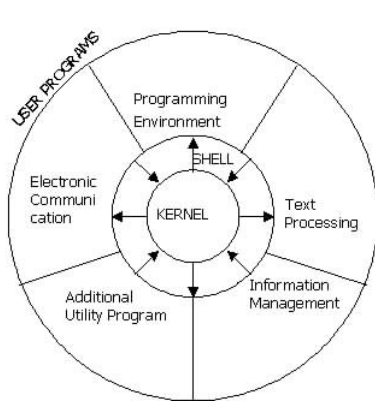


Layering: Διαστρωμάτωση



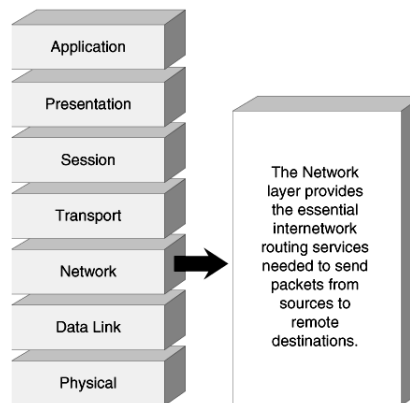
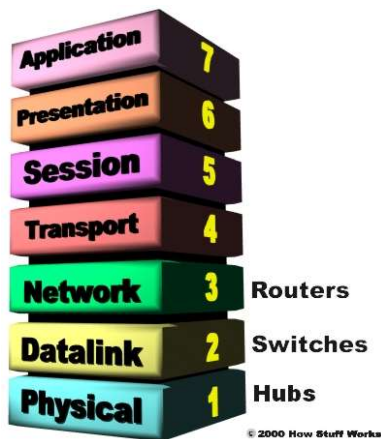
Διαστρωματωμένα Συστήματα: Παραδείγματα Examples of Layered Systems

The Unix Operating System



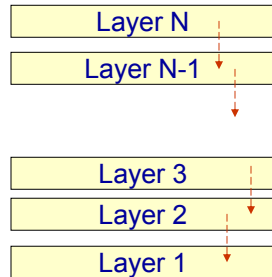
Διαστρωματωμένα Συστήματα: Παραδείγματα Examples of Layered Systems

OSI Network Protocol



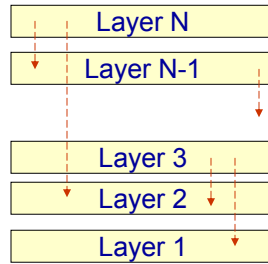


Διαστρωματωμένη Αρχιτεκτονική: Κλειστή vs Ανοικτή Layered Architectures: Closed vs Open



Closed

- each layer can use services of the immediately lower layer
- minimizes dependencies



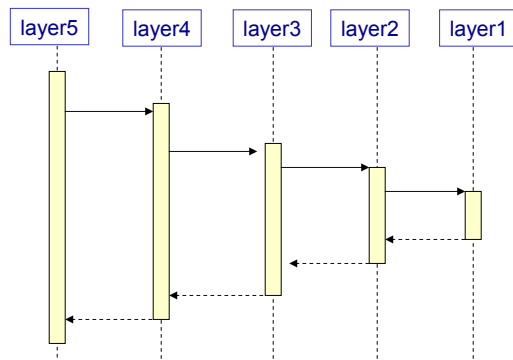
Open

- each layer can use services of any lower layer
- increased dependencies however the code can be more compact

Recall the trade-off between understandability and efficiency: increasing the understandability of a design usually results in inefficiencies, while focusing only on efficiency usually results in design that is difficult to understand by someone else



Ενδεικτικό Διάγραμμα Ακολουθίας μιας κλειστής διαστρωματωμένης αρχιτεκτονικής The form of sequence diagrams in a closed layered architecture





Παράδειγμα υλοποίησης μιας Κλειστής Διαστρωματωμένης Αρχιτεκτονικής Example of implementing a Closed Layered Architecture

```
public abstract class L1Provider {
    public abstract void L1Service();
}

public abstract class L2Provider {
    protected L1Provider level1;

    public abstract void L2Service();
    public void setLowerLayer(L1Provider l1)
    {
        level1 = l1;
    }
}

public abstract class L3Provider {
    protected L2Provider level2;

    public abstract void L3Service();
    public void setLowerLayer(L2Provider l2)
    {
        level2 = l2;
    }
}
```

```
public class DataLink extends L1Provider {
    public void L1Service() {
        println("L1Service doing its job");
    }
}

public class Transport extends L2Provider
{
    public void L2Service() {
        println("L2Service starting its job");
        level1.L1Service();
        println("L2Service finishing its job");
    }
}

public class Session extends L3Provider{
    public void L3Service() {
        println("L3Service starting its job");
        level2.L2Service();
        println("L3Service finishing its job");
    }
}
```

```
public class Network {
    public static void main(String args[]) {
        DataLink dataLink = new DataLink();
        Transport transport = new Transport();
        Session session = new Session();

        transport.setLowerLayer(dataLink);
        session.setLowerLayer(transport);

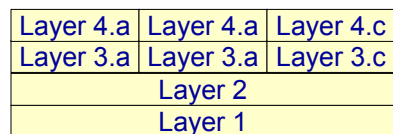
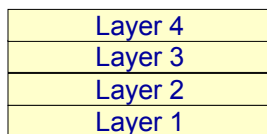
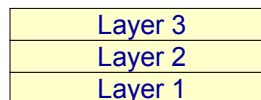
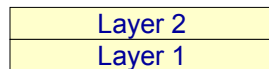
        session.L3Service();
    }
}
```

EXECUTION RESULT:

```
L3Service starting its job
L2Service starting its job
L1Service doing its job
L2Service finishing its job
L3Service finishing its job
```



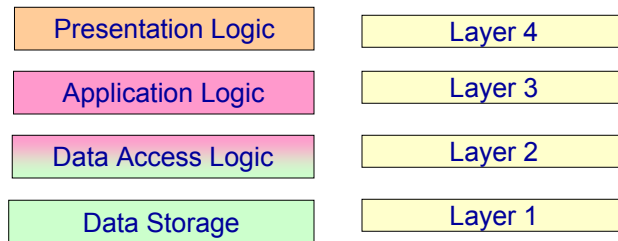
Πλήθος στρωμάτων Number of Layers



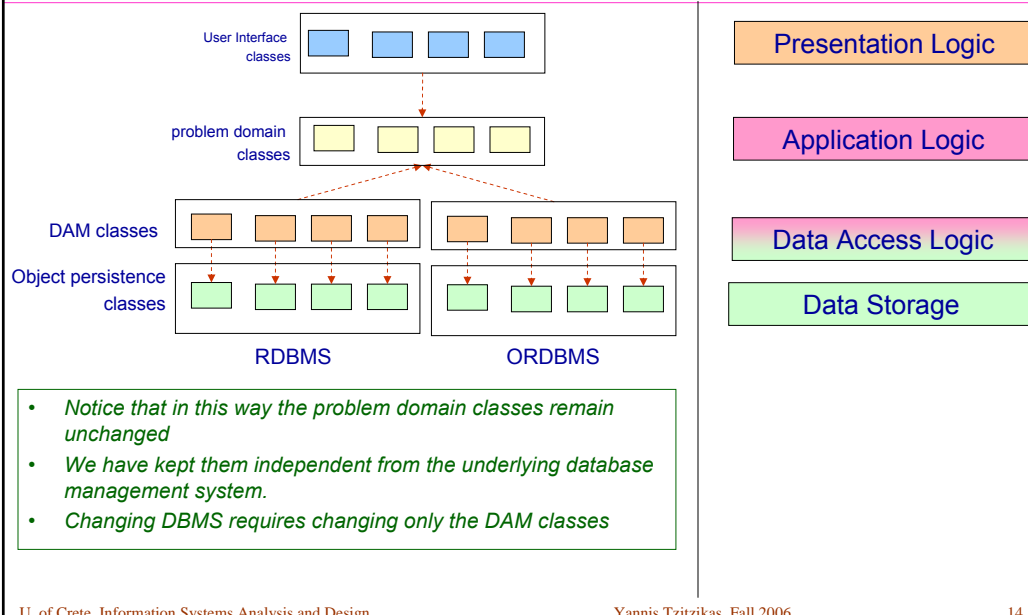


Θεωρώντας τα 4 βασικά συστατικά λογισμικού ενός ΠΣ ως στρώματα Considering the 4 primary software components of an IS as Layers

- Presentation logic
- Application logic
- Data access logic
- Data storage




Υπενθυμιστικό: Σχεδίαση Στρώσης Διαχείρισης Δεδομένων Refresher: Data Mgmt Layer Design








Τα 3 κυριότερα εξαρτήματα υλικού ενός συστήματος The 3 primary hardware components of a system

- Client computers 
- Servers 
- Network 



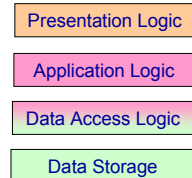
Τύποι Αρχιτεκτονικών Kinds of Architectures

Primary Hardware components

- Client computers 
- Servers 
- Network 

X

Primary Software components



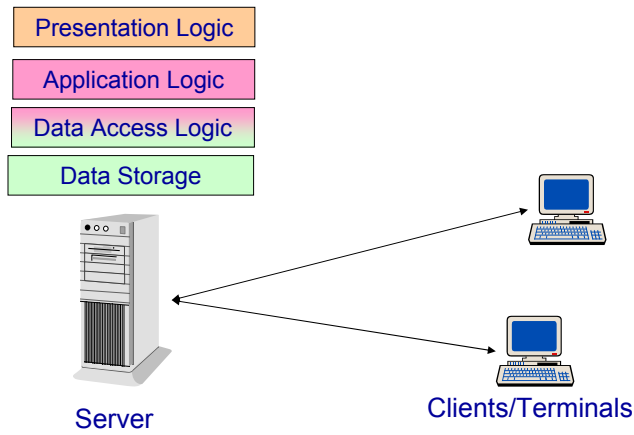
= architectures

According to the distribution of the 4 basic layers to hardware nodes we can distinguish the following architectures:

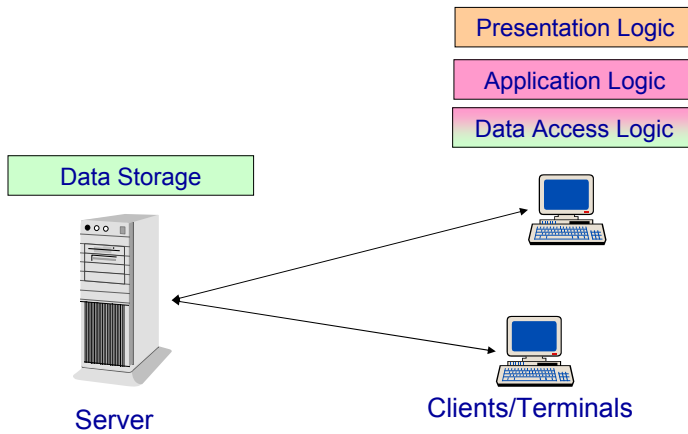
- (a) **Server-based** computing
- (b) **Client-based** computing
- (c) **Client-server**-based computing
- (d) **3/4/N tiers** computing



(a) Server-based Computing

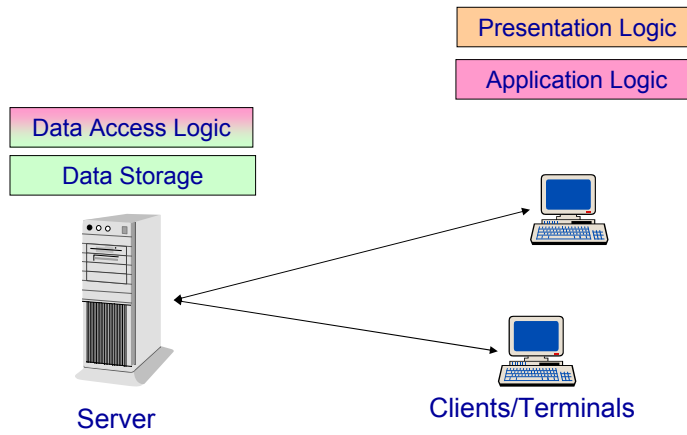


(b) Client-based Computing

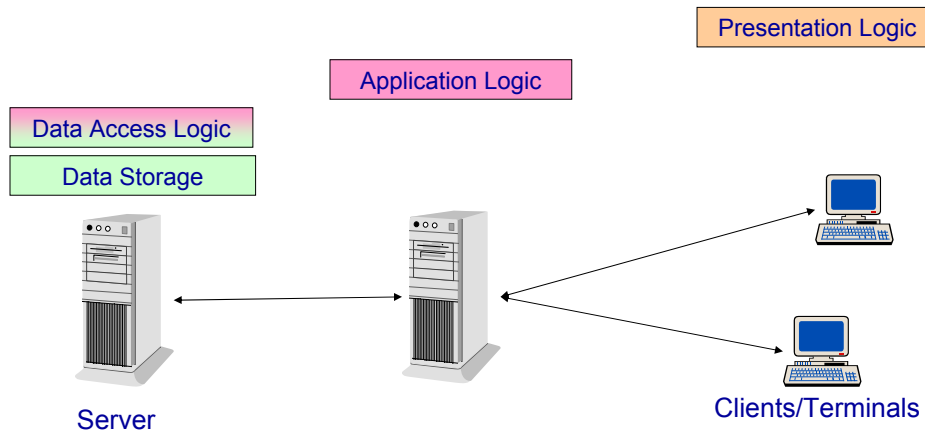




(c) Client-Server-based Computing (2 Tiers)

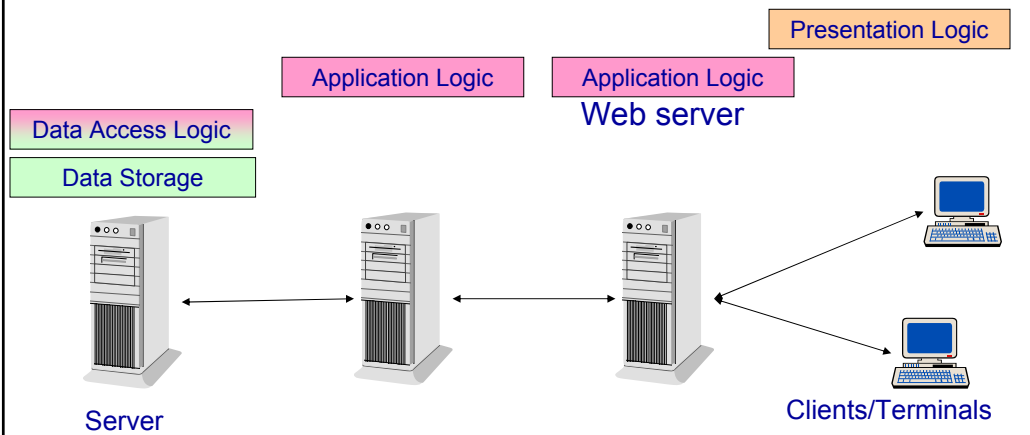


(d) 3 tiers based computing





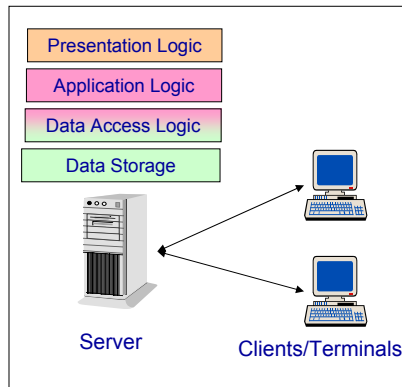
(d') 4 tiers based computing



Some more details about the previous architectures



(a) Server-based Computing

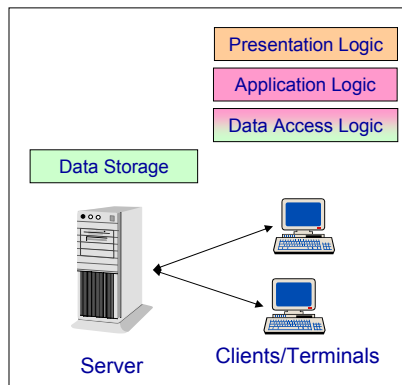


Characteristics

- The server does almost everything. The client is actually a very thin client
- [-]: The Server has very high load
 - the clients do not contribute to the computation
- [+]: Not so difficult to implement
- [+] If platform changes (e.g. OS) we have to rewrite only the thin client



(b) Client-based Computing

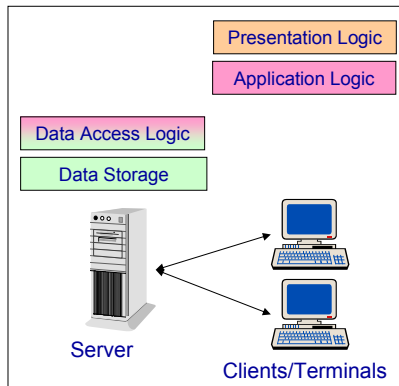


Characteristics

- [+] The server has less load
- [-] The clients are very heavy (they should be computationally powerful machines)
- [-] sometimes a lot of data have to be communicated through the network
- [-] If we the OS changes then we have to rewrite the 3 layers of the client
 - (in server-based computing we could keep the server running in the old OS) and we would need to change only the thin client so that to run in the new OS



(c) Client-Server-based Computing (2 Tiers)



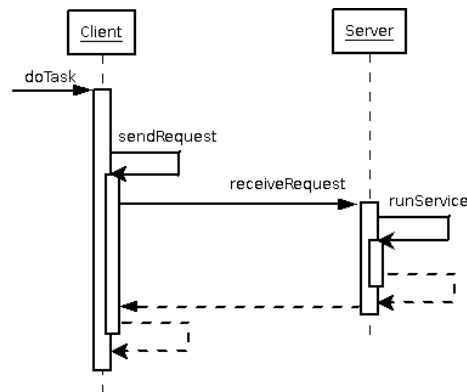
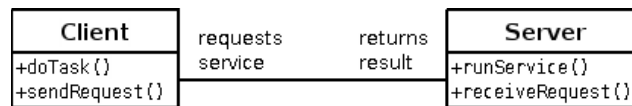
Characteristics

This is like having a **thick client** (thin client: if responsible only for the UI)

- [+] The client has less load (comparing to client-based computing)
- [+] The server has less load comparing to server-based computing
- [-] We have to rewrite the application logic if platform changes
- [-] Sometimes a lot of data have to be communicated through the network
- [+] Good overall performance

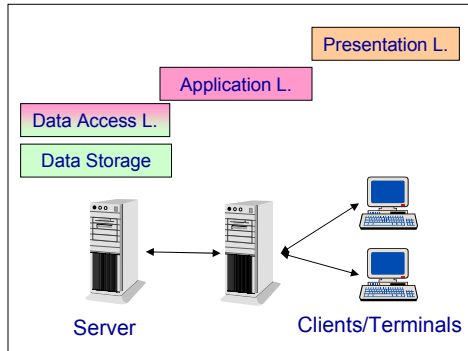


Client Server: Class and Interaction Diagrams





(d) 3 tiers based computing

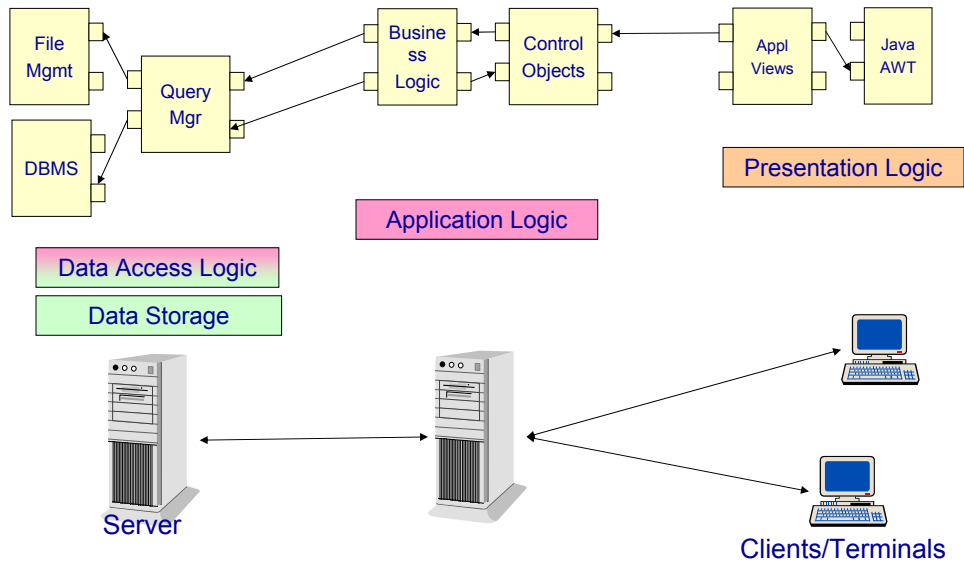


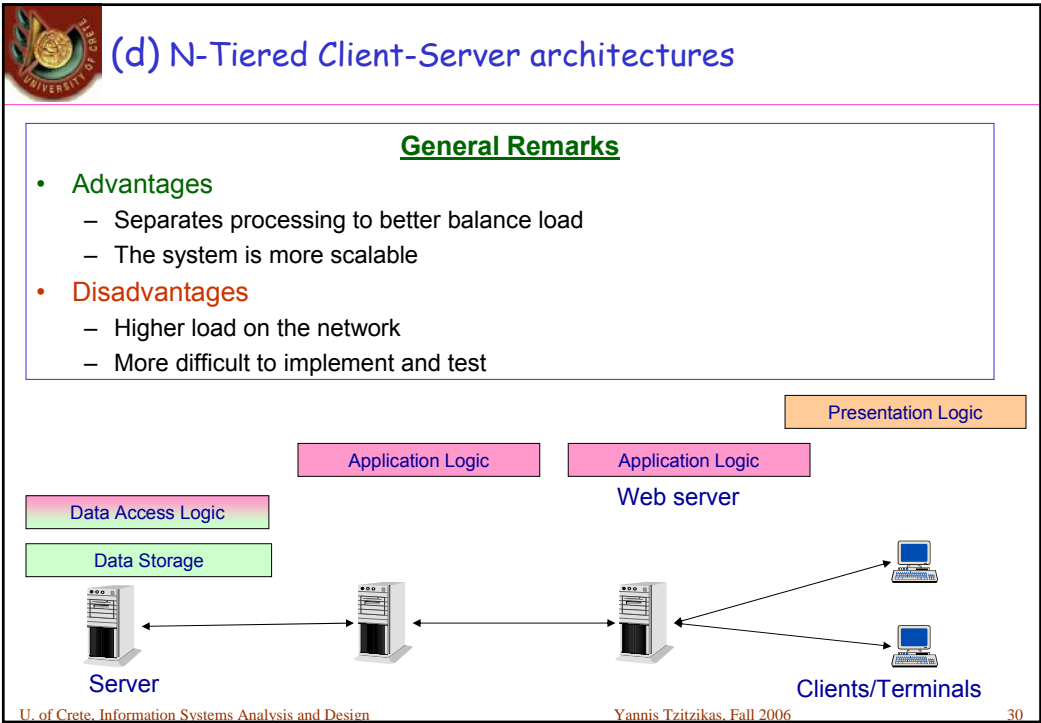
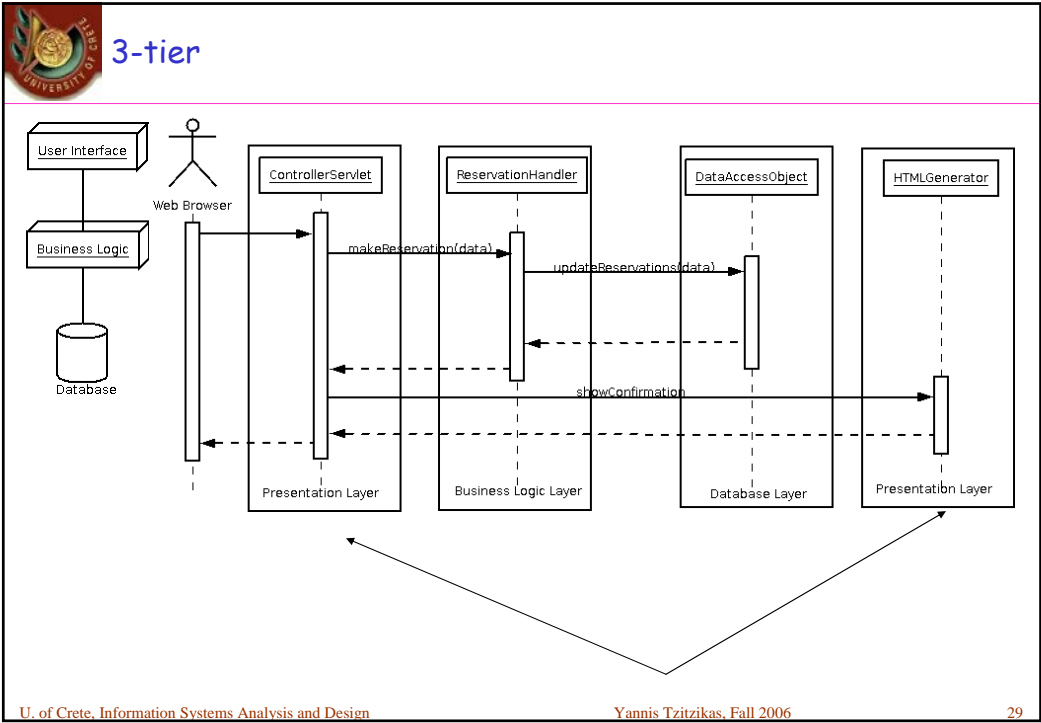
Characteristics

- [+] Good load balancing
the server and the client have less load
- [+] the UI component is independent of the rest system
 - server-based computing has also this property but in that case the server has excessive load
 - This architecture is suited for heterogeneous environments
- [-] more complex implementation - more data are transferred through the network



(d) 3 tiers: Example







Επιλέγοντας Αρχιτεκτονική Λογισμικού Selecting a Computing Architecture

	Server-Based	Client-based	Client-server
Cost of infrastructure	Very high	Medium	Low
Cost of development	Medium	Low	High
Ease of development	Low	High	Low-medium
Interface capabilities	Low	High	High
Control and security	High	Low	Medium
Scalability	Low	Medium	High

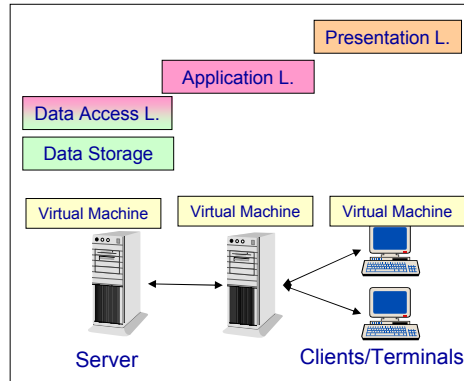


Virtual Machine



Εικονική Μηχανή Virtual Machine

- It is a form of layered architecture
- It allows using the same API independently of the underlying OS/hardware
- The compiler produces intermediate code (bytecodes in Java) which can be handled by the virtual machine



Service-Oriented Computing Υπηρεσιοστρεφής Υπολογισμός



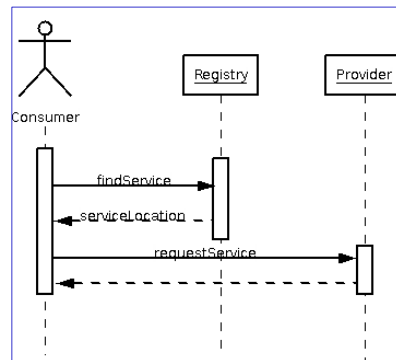
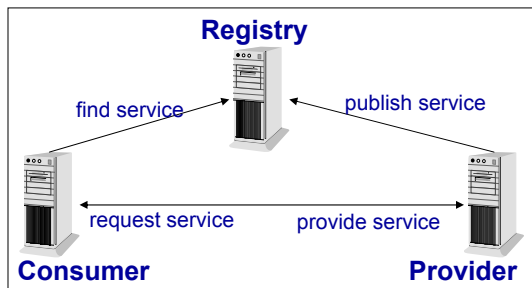
Υπηρεσιοστρεφής Υπολογισμός Service-oriented Computing

SOA: Service Oriented Architecture

Software is considered as a set of services

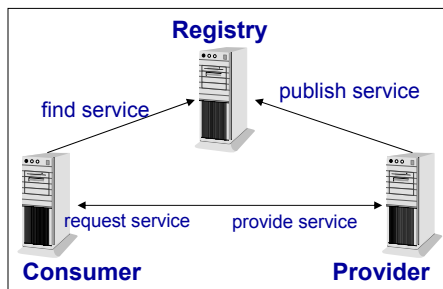
We can have

- **service providers**
- **consumers**
- **registries** (catalogs of available services)



Υπηρεσιοστρεφής Υπολογισμός (2) Service-oriented Computing (2)

- Based on open standards (SOAP, REST, WSDL, UDDI)
- Data is exchanged using XML



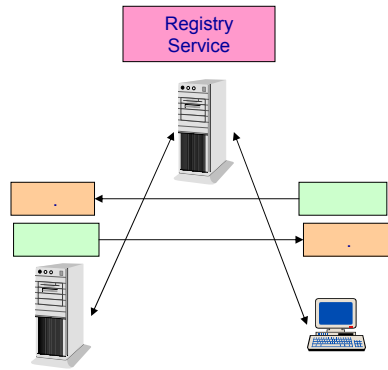
Characteristics

- [+] complete separation between providers & consumers
- [+] the same service can be provided with different characteristics (quality, price, speed, etc) from different providers => competitiveness
- [+] open standards
- [-] not mature technology, no registries for business services

- **Web Services**
 - Data are exchanged in XML (SOAP, REST)
 - Data are transferred using HTTP
 - The “interface” provider-consumer is described in XML (WSDL)



Υπηρεσιοστρεφής Υπολογισμός (3) Service-oriented computing (3)



Extra Reading

- [1] “Model-driven Web Services Development”
– <http://folk.uio.no/roygr/EEE-2004.pdf>
- [2] “From UML to BPEL (*Model Driven Architecture in a Web services world*)”
– <http://www-128.ibm.com/developerworks/websevi ces/library/ws-uml2bpel/>



Service-oriented computing and UML (Based on [1])

- With UML we can express the contents and behavior of web services in a more understandable way than WSDL

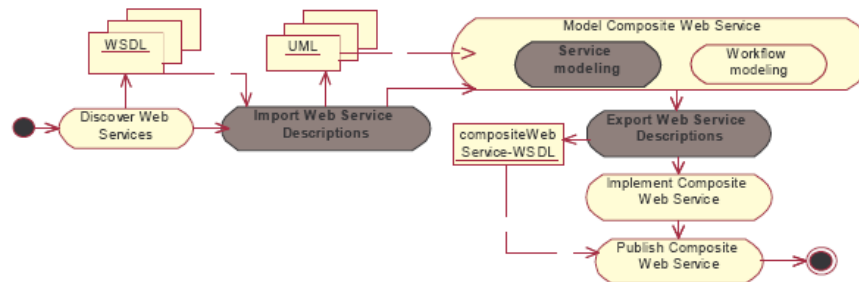
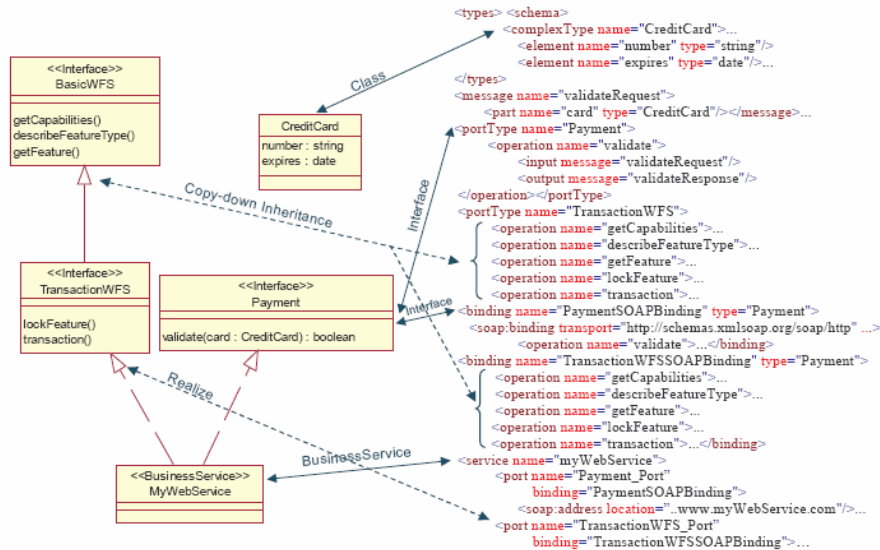


Figure 1: Steps of model-driven web services development



Service-oriented computing and UML (Based on [1])



U. c

Figure 2: Conversion between a UML model and a WSDL document

39



From UML to BPEL (Based on [2])

- The Business Process Execution Language for Web Services (BPEL4WS or BPEL for short) is an XML-based standard for defining how you can combine Web services to implement business processes. It builds upon the Web Services Definition Language (WSDL) and XML Schema Definition (XSD).
- [2] describes a tool which takes processes defined in the Unified Modeling Language (UML) and generates the corresponding BPEL and WSDL files to implement that process.
- A UML Profile is used for defining stereotypes relating to Business Process Execution Language for Web Services. A Mapping is provided for automatically generating Web services artifacts (BPEL, WSDL, XSD) from a UML model meeting the profile.

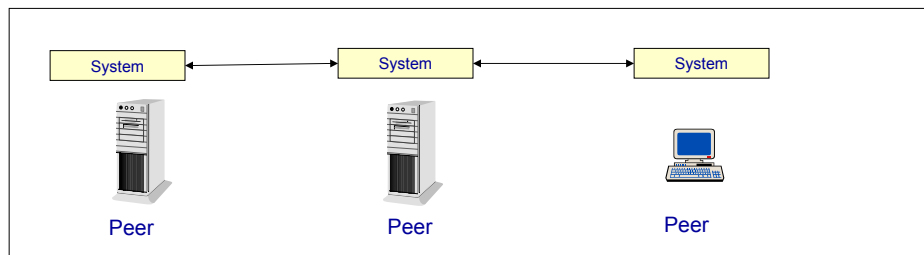


Διομότιμες Αρχιτεκτονικές Peer-to-Peer (P2P) architectures



Peer to Peer

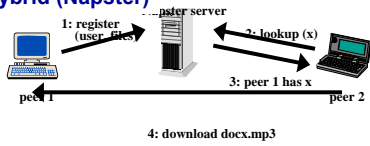
- Pure
 - all are equal. No layering. Each peer depends on the others



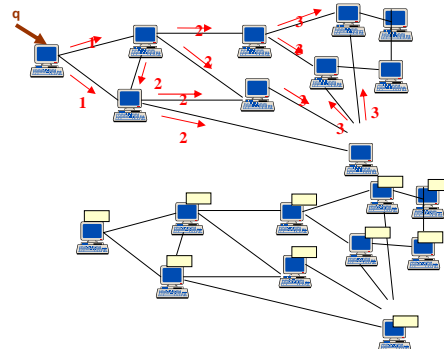


Peer-to-Peer Architectures

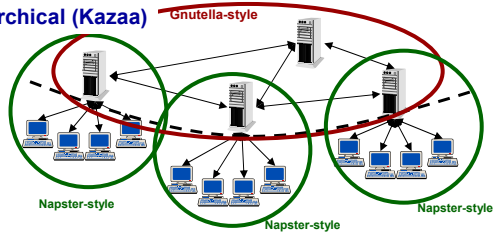
Hybrid (Napster)



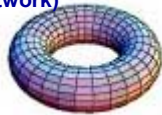
Decentralized (Gnutella)



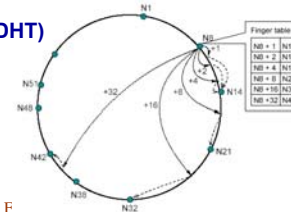
Hierarchical (Kazaa)



CAN (Content Addressable Network)



Chord (DHT)



U. of Crete, Information Systems Analysis and Design

Yannis Tzitzikas, F



Πρωτόκολλα Επικοινωνίας Communication Protocols



How objects of different layers at different machines can communicate ?

- **RPC (Remote Procedure Call):**
 - can invoke a remote procedure, send results, (RPC is widely supported in languages such as C, C++)
- **RMI (Remote Method Invocation)**
 - in java (recall www.csd.uoc.gr/~hy252)
- **DCOM**
 - Microsoft's Distributed Component Object Model
- **CORBA (Common Object Request Broker Architecture)**
 - The object-oriented industry standard by OMG (1995)
- **SOAP (Simple Object Access Protocol)**
 - uses XML to encapsulate messages and data that can be sent from one process to another



- **RMI or DCOM are language/operating system specific protocols**
 - they restrict the design to implementation on certain platforms
- **CORBA or SOAP are open standards**
 - they allow building component-based systems that are not tied to particular platforms



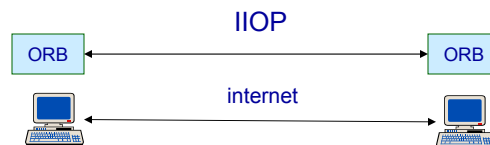
Case: CORBA

- CORBA separates the interface of a class (the operations it can carry out) from the implementation of that class.
- The interface can be compiled into a program running on one computer.
- An object instance can be created or accessed by name.
- To the client program it appears to be in memory on the same machine, however, it may actually be running on another computer.
- When the client program sends it a message to invoke one of its operations, the message and its parameters are converted into a format that can be sent over the network (known as *marshalling*). At the other end the server unmarshals the data back into a message and parameters and passes it to the implementation of the target object.
- This object then carries out the operation and, if it returns a value, that value is marshalled on the server, unmarshalled on the client and finally provided as a return value to the client program



CORBA (2)

- CORBA achieves this by means of programs known as ORBs (Object Request Brokers) that run on each machine.
- The ORBs communicate with each other by means of an Inter-ORB Protocol (IOP).
- Over the Internet, the protocol used is IIOP (Internet IOP).





CORBA (3)

- To use this facility, the developer must specify the interface (public attributes and operations) of each class in an **Interface Definition Language (IDL)**.
- The IDL file is then processed by a program that converts the interface to a series of files in the target language or languages.
- In Java, this program is called IDL2JAVA and produces
 - a file that defines the interface in Java,
 - a stub file that provides the link between the client program and the ORB,
 - it implements the interface on the client and is compiled into the client program
 - a file that provides a skeleton for the implementation of the server
 - it implements the interface on the server; the developer updates this file (provides the implementation) and it is compiled on the host

The IDL file for a class Location

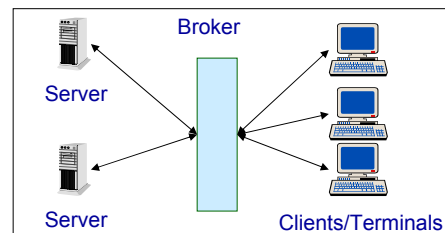
```
Module CretanTourismApplication
{ interface Location
  { attribute string locationCode;
    attribute string locationName;
    void addHotel(in Hotel hotel);
    void removeHotel(in string hotelCode);
    int numberOfHotels(); };
};
```



CORBA (4)

Supporting different PLs, Wrapping legacy systems

- CORBA is known as **middleware**, as it acts as an intermediary between clients and servers. As such it enables the implementation of a 3 or 4 tier architecture that isolates the UI and client programs from the implementation of classes on one or more servers.



- CORBA also provides **interoperability between different languages**: a Java client program can invoke operations on a C++ object that exist on a separate machine.
- CORBA also makes it possible to encapsulate pre-existing programs (legacy systems) written in non-object oriented languages by **wrapping them in an interface**. To the client it looks like an object, but internally it may be implemented in a language like COBOL.



CORBA (5) More advanced features

Systems developed using CORBA can be set up so that the remote objects are located on a named machine and accessed by name. This is what we need in the majority of applications.

CORBA also provides a number of more advanced services:

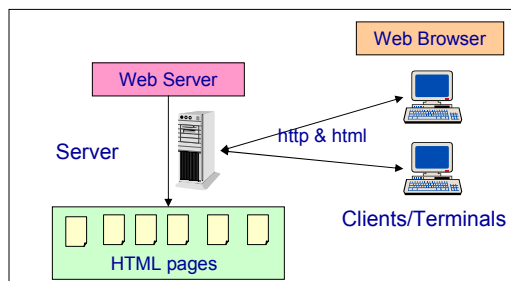
- Services for locating objects by name when it is not known where they are running.
- Services for locating objects that implement a certain interface and for interrogating an object to determine its interface (operations, parameter types and return types) in order to dynamically invoke its operations.



Ιστο-βασισμένες εφαρμογές Web-based applications

HTTP (HyperText Transfer Protocol): transfers hypertext documents over the internet

- HTML (HyperText Markup Language): defines hypertext documents

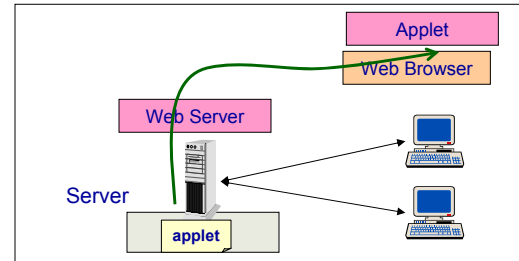
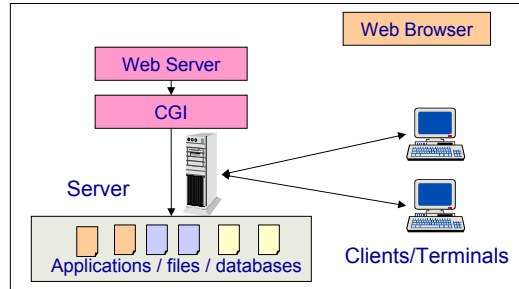


Very static architecture



Web-based applications: Adding .. "dynamism"

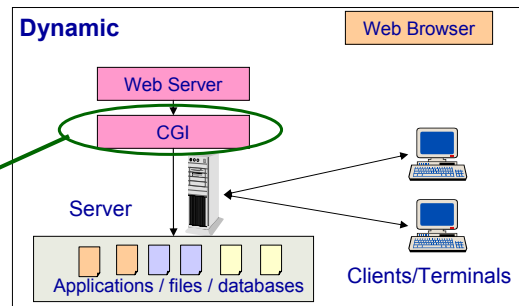
CGI (Common Gateway Interface):
CGI scripts are programs (e.g. a unix shell script or a perl script) that reside on the web server and can be invoked by elements of the web pages



Web-based applications

alternatives

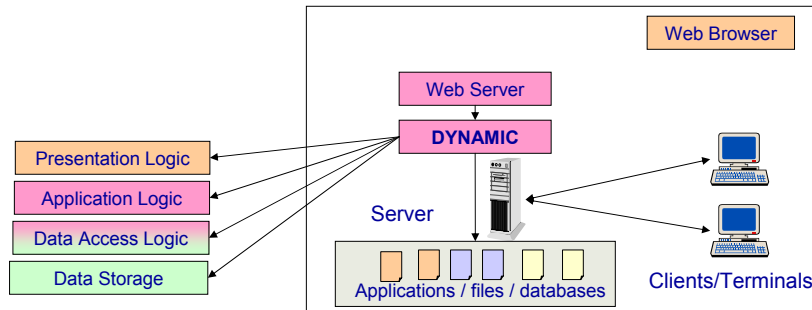
- **ASP (Active Server Pages)**
 - limited to Microsoft Platform
- **JSP (Java Server Pages)**
 - JSP is designed to be platform and server independent, created from a broader community of tool, server, and database vendors





Web-based applications

- Here we have to design our layers assuming the Web platform

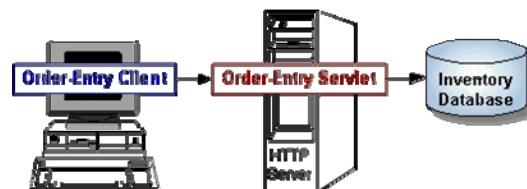


- So Web Servers and the Web Browsers become parts of our information system.



Servlets

- **Servlets are to servers what applets are to browsers.**
- Servlets are modules that extend request/response-oriented servers, such as Java-enabled web servers.
 - A servlet might be responsible for taking data in an HTML order-entry form and applying the business logic used to update a company's order database.
- Servlets can be embedded in many different servers because the servlet API, which you use to write servlets, assumes nothing about the server's environment or protocol. Servlets have become most widely used within HTTP servers; many web servers support Java Servlet technology.





Servlets (II)

Servlets are an effective replacement for CGI scripts.

- They are easier to write and run faster

So we can use servlets to handle HTTP client requests.

- We can have servlets to process data POSTed over HTTPS using an HTML form, including purchase order or credit card data. A servlet like this could be part of an order-entry and processing system, working with product and inventory databases, and perhaps an on-line payment system.

Other Uses for Servlets

- A servlet can handle multiple requests concurrently, and can synchronize requests. This allows servlets to support systems such as on-line conferencing.
- Servlets can forward requests to other servers and servlets. Thus servlets can be used to balance load among several servers that mirror the same content, and to partition a single logical service over several servers, according to task type or organizational boundaries.



A Simple Servlet (Hello World)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

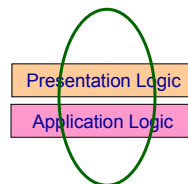
public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse
        response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```



Σχεδιαστικό Μοτίβο Model-View-Controller (MVC)

Pattern Model-View-Controller (MVC)



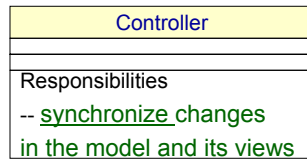
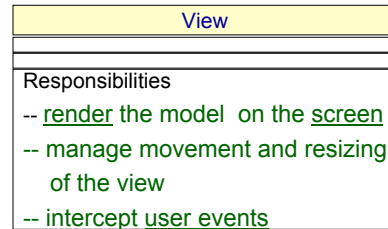
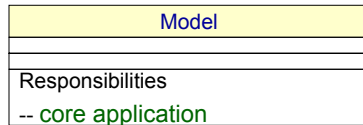
Model-View-Controller (MVC)

- This pattern is used in applications where the UI is very important
- Motivation
 - same data may be displayed differently
 - display and application must reflect data changes immediately
 - UI changes should be easy and even possible at runtime
 - Porting the UI to another platform should not affect core application code
- Solution
 - Divide application into 3 parts
 - Model
 - View
 - Controller



Model-View-Controller

Model: provides the essential functionality of the application (application logic)
View: supports a particular style of interaction with the user (display output)
Controller: accepts user input in the form of events and synchronizes changes between the model and its views (user input)

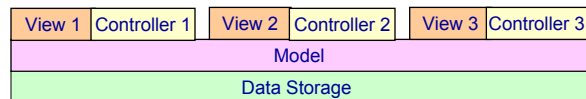
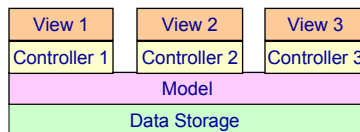
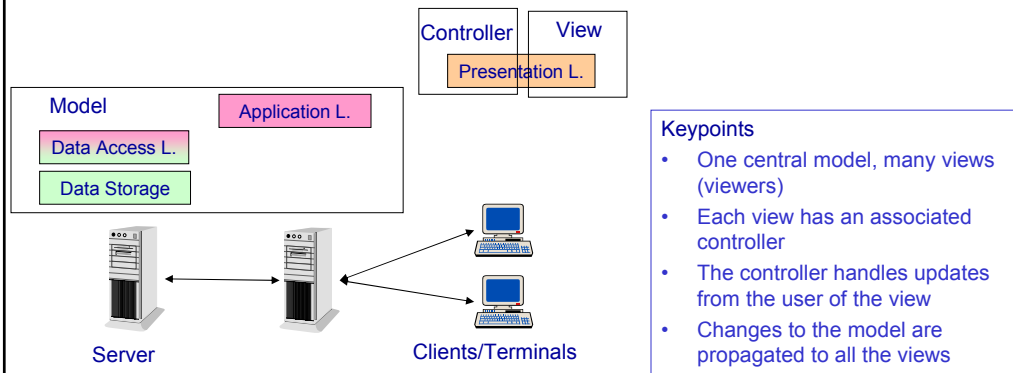


Decoupling achieved: We can:

- have multiple views/controllers for the same model
- reuse views/controllers for other models

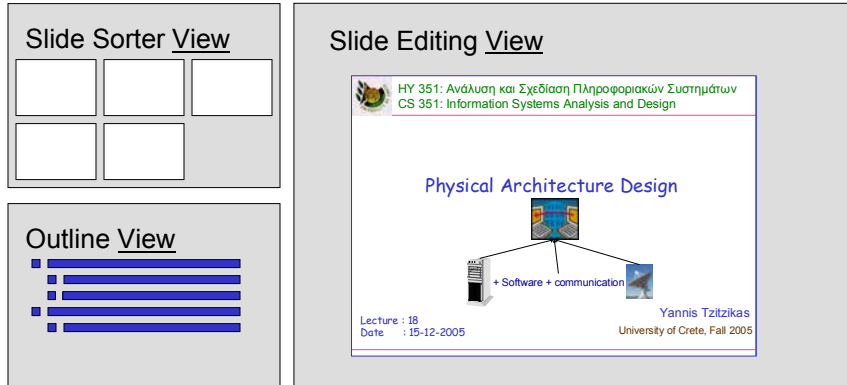


MVC: connection with the previous discussion





Example: The Views of Powerpoint

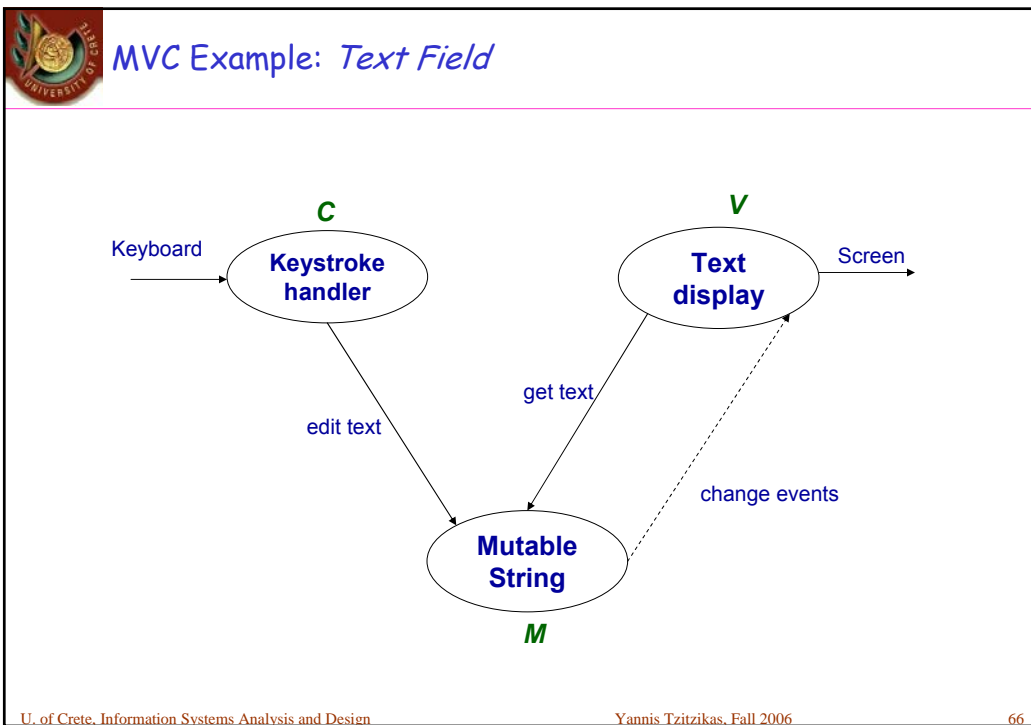
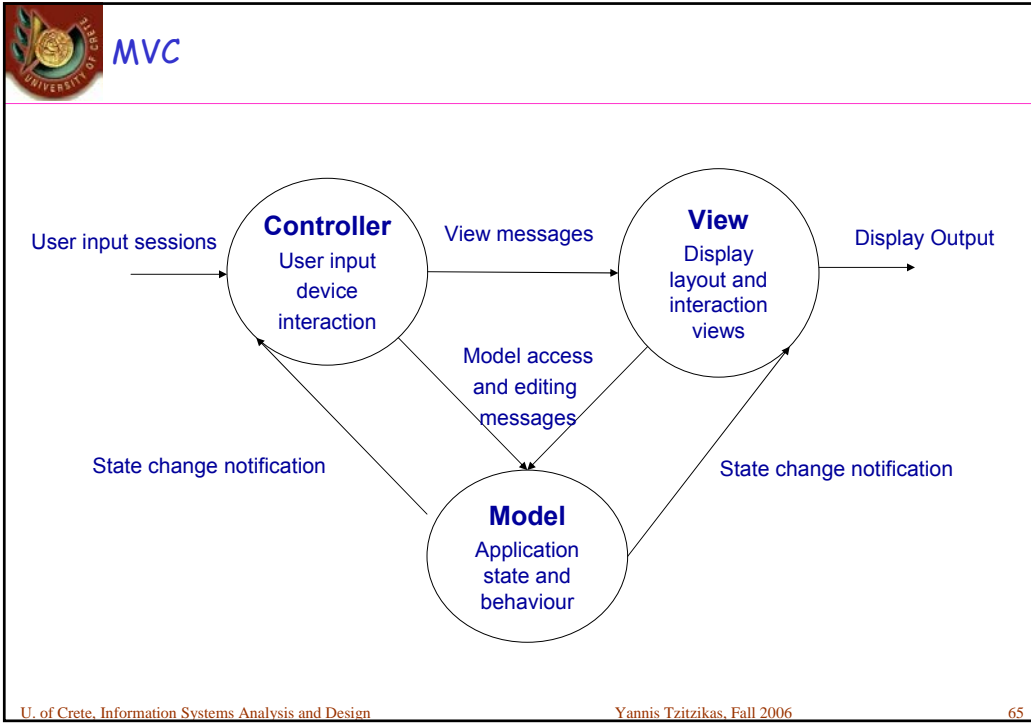


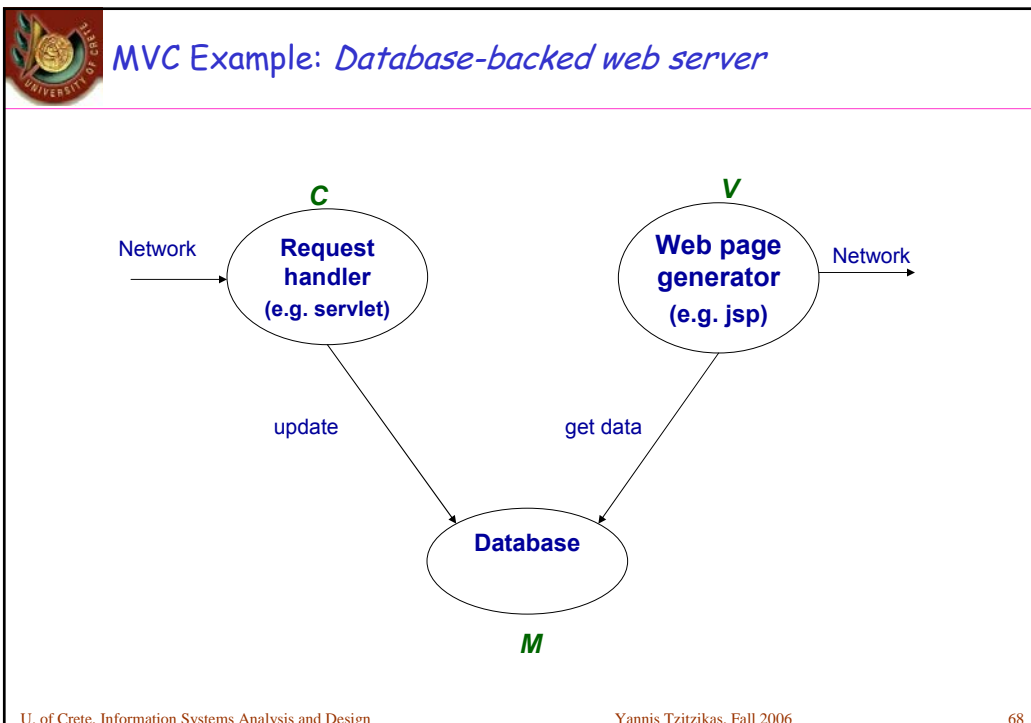
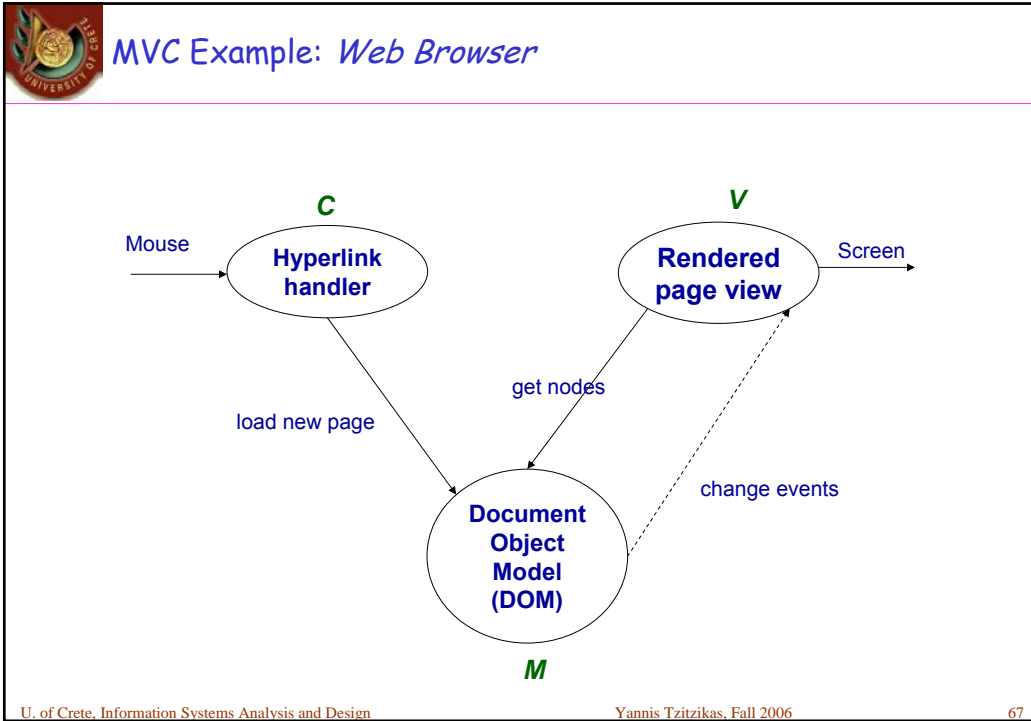
The structure of the model of Powerpoint



MVC

- **Model**
 - Core application code
 - maintains application state
 - Contains a list of observers (view or controller)
 - Has a broadcast mechanism to inform views of a change
- **View**
 - displays information to user
 - obtains data from model
 - each view has a controller
- **Controller**
 - handles input from user as events (keystrokes, mouse clicks and movements)
 - maps each event to proper action on model and/or view



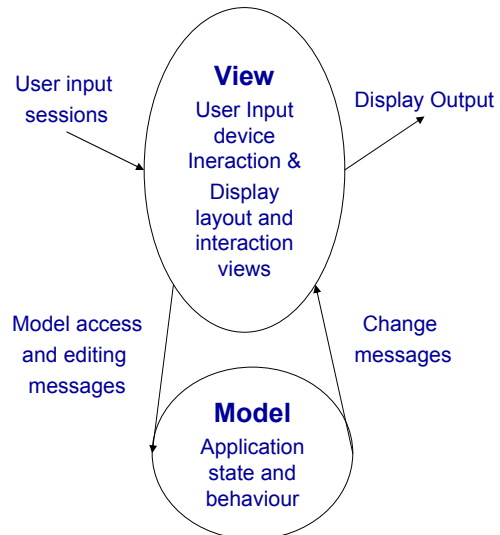




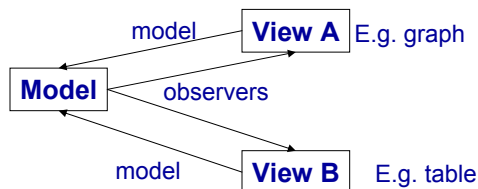
MVC and MV

In many cases, view and controller are very tightly coupled.

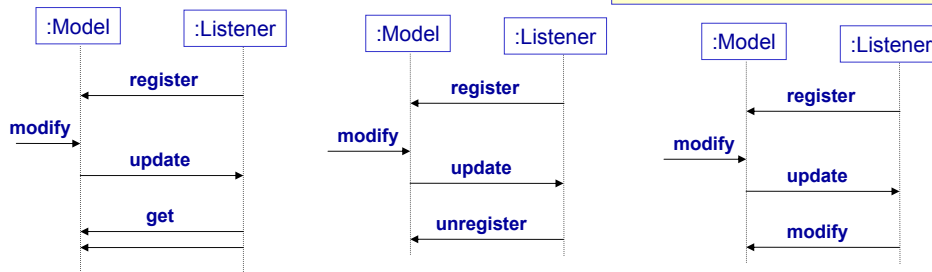
- so instead of MVC we have MV (Model-View)
- a reusable view manages both output and input
 - also called widgets, components, ...
 - e.g. scrollbars, buttons, ...



Observer pattern is used to decouple model from views



```
interface Model {
    void register(Observer)
    void unregister(Observer)
    Object get()
    void modify()
}
interface Observer {
    void update(Event)
}
```





- *How we can depict the Physical Architecture of a System?*

- *Is there any standard diagrammatic notation?*

=> UML Components and Deployment diagrams

– (next lecture)



Reading and References

- **Systems Analysis and Design with UML Version 2.0** (2nd edition) by A. Dennis, B. Haley Wixom, D. Tegarden, Wiley, 2005. Chapter 13
- **Object-Oriented Systems Analysis and Design Using UML** (2nd edition) by S. Bennett, S. McRobb, R. Farmer, McGraw Hil, 2002, Chapter 18
- **The Unified Modeling Language User Guide** (2nd edition) by G. Booch, J. Rumbaugh, I. Jacobson, Addison Wesley, 2004
- Slides of: UI Software Architecture, 6.831 (UI Design and Implementation)



HY 351: Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων CS 351: Information Systems Analysis and Design

HY351:
Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων
Information Systems Analysis and Design



Physical (or Implementation) Diagrams

- UML component diagrams
- UML deployment diagrams



Γιάννης Τζιτζίκας

Διάλεξη : 18
Ημερομηνία : 24-2-2007
Θέμα :



Βασικές ερωτήσεις Key questions

Η υπολογιστικές πλατφόρμες αποτελούνται από υλικό, λογισμικό (PLs, DBMSs) και δικτύωση

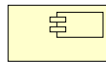
Ποια πλατφόρμα είναι πιο κατάλληλη για αυτό το πληροφοριακό σύστημα;

- Πώς να επιλέξουμε το υλικό (hardware);
- Πώς να επιλέξουμε το λογισμικό (software);
- Πώς να επιλέξουμε τη δικτύωση (networking);
- Πώς να εκφράσουμε τη φυσική αρχιτεκτονική (μάθημα 18) του συστήματος με μια σπάνταρτ διαγραμματική μορφή;





Διαγράμματα Εξαρτημάτων Component Diagrams

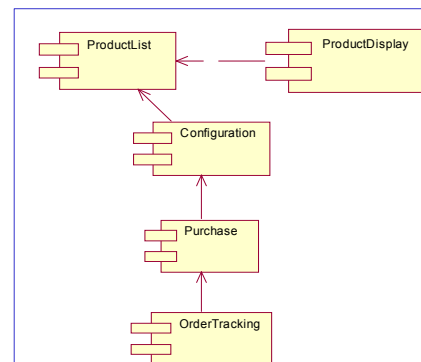
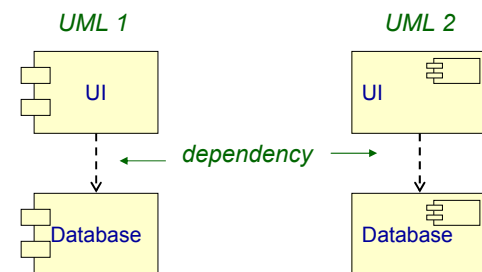


Component Diagrams (διαγράμματα εξαρτημάτων)

Component Diagrams show various components and their dependencies

- **Component:**
 - physical module of code (like package, class, or even file)
- **dependency:**
 - change dependency (e.g. communication dependencies, compilation dependencies)

Συμβολισμοί :





Τα χαρακτηριστικά ενός εξαρτήματος The Characteristics of a Component

- a unit of independent deployment (never deployed partially)
 - sufficiently documented and self-contained to be “plugged into” other components by a third-party
 - it cannot be distinguished from copies of its own; in any given application, there will be at most one copy of a particular component
 - it is a replaceable part of a system (can be replaced by another component that conforms to the same interface)
 - it fulfils a clear function and is logically and physically cohesive
 - it may be nested in other components
- [Szyperski 98, Rumbaugh et al. 99, Maciaszek 2005]



Εξαρτήματα Components

- Components are like classes and packages
 - can be connected through interfaces
- Components are about how customers want to relate to software
 - they want to be able to upgrade it like they can upgrade their stereo (in pieces)
 - they want to mix and match pieces from various manufacturers
 - reasonable but difficult to satisfy
- So we could define a component as:
 - a logical and replaceable part of a system that conforms to and provides the realization of a set of interfaces
 - an independently purchasable and upgradeable piece of software

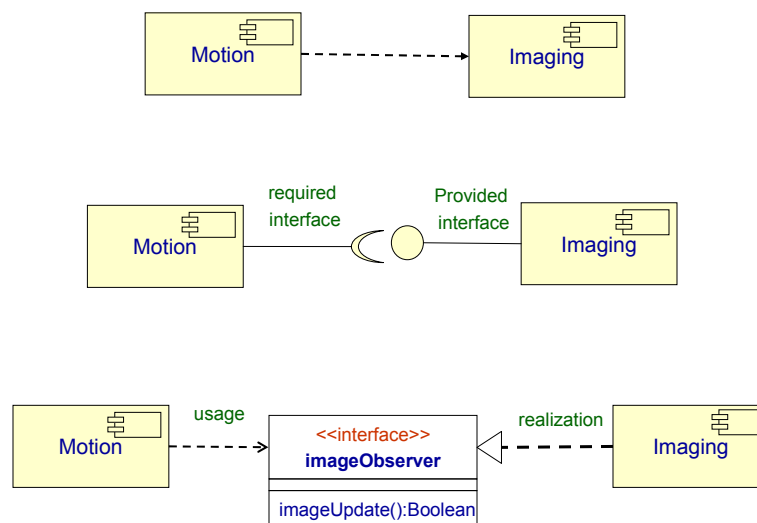


Εξαρτήματα και σχετικές έννοιες Components and related Notions

- **Component**
 - a replaceable part of a system that conforms to and provides the realization of a set of interfaces
- **Interface:**
 - a collection of operations that specify a service that is provided by or requested from a class or component
- **Port**
 - a specific window into an encapsulated component accepting messages to and from the component conforming to specified interfaces
- **Part**
 - (an internal component) the specification of a role that composes part of the implementation of a component.
- **Internal structure**
 - the implementation of a component by means of a set of parts that are connected together in a specific way
- **Connector:**
 - a communication relationship between two parts or ports within the context of component

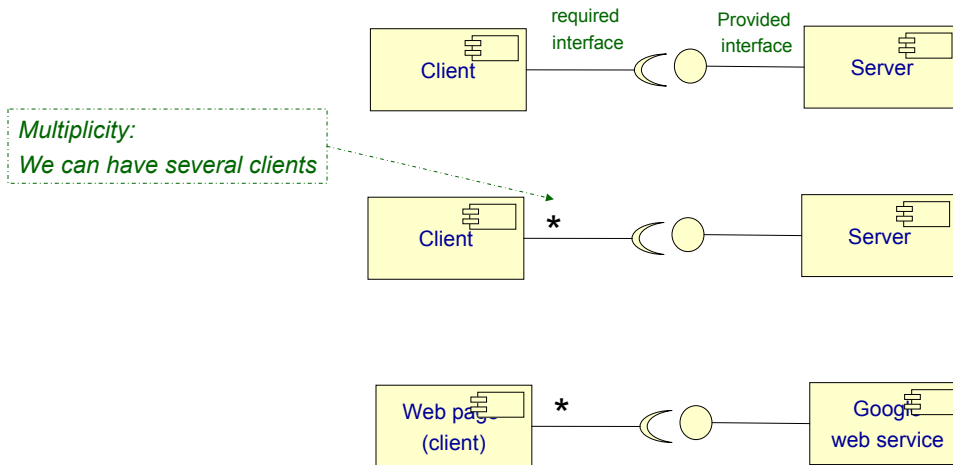


Εξαρτήματα και Διεπαφές Components and interfaces



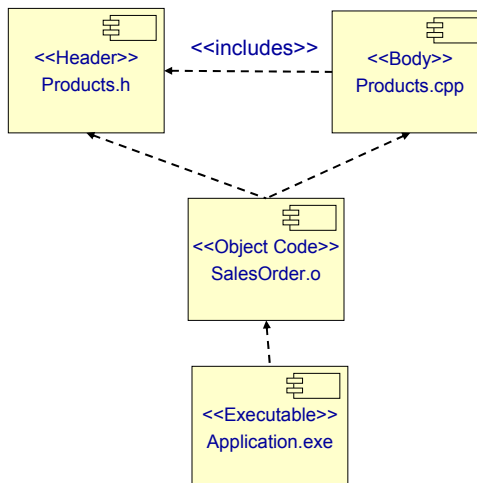


Εξαρτήματα και Διεπαφές (II) Components and interfaces (II)



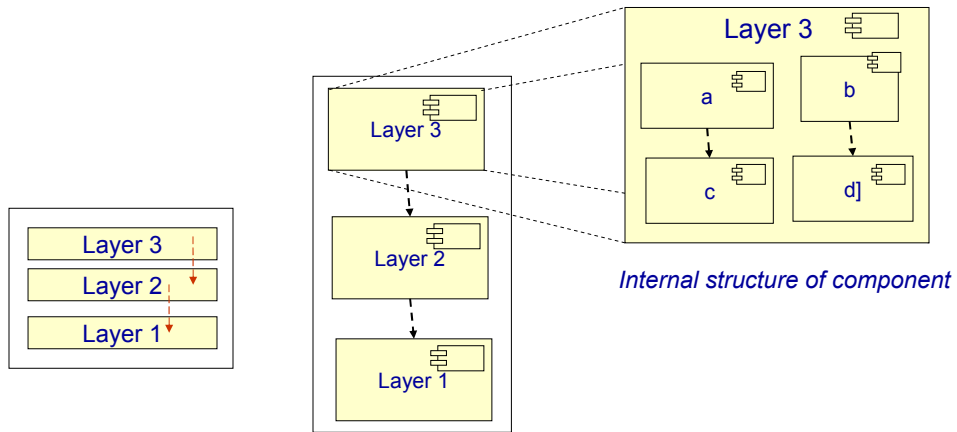
Παράδειγματα εξαρτημάτων Fine-grained Components: Example

We could use component diagrams for modeling more fine-grained components (e.g. files).

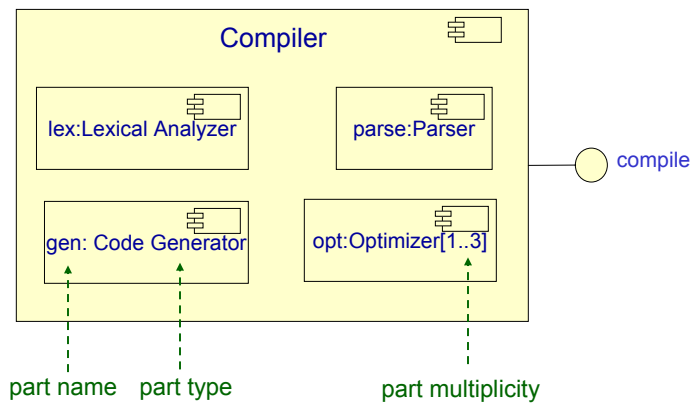




Coarse-grained components: e.g. Layers



Εσωτερική Δομή Εξαρτημάτων Internal Structure of Components





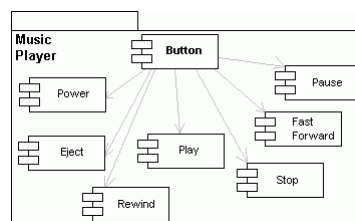
Παράδειγμα

πηγή: <http://odl-skopje.etf.ukim.edu.mk/uml-help/>

- Suppose that we need to build up a software for playing a music from a CD-ROM Drive. A visual programming language might be used (VisualBasic or Delphi for example). If language supports multimedia controls, than we can use its components an reprogramm them if necessary, or we can programm new components. One possible graphical design for our player might be:



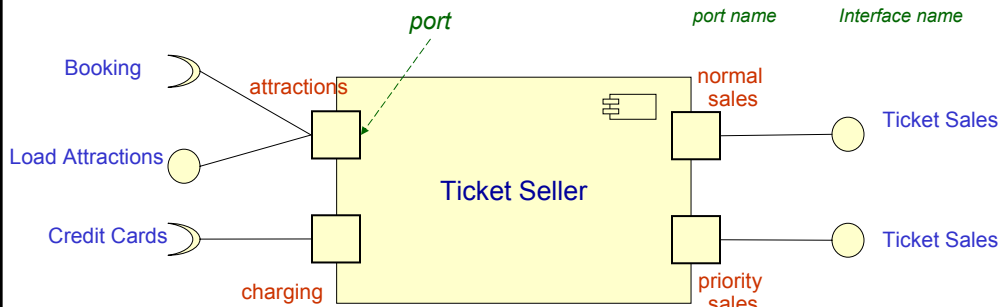
- As you can see this UML Music Player needs these controls:
- play stop eject pause fast forward rewind power
- These controls will be realized by **buttons**, thus we'll have a button performing these controls. If we look at buttons as separete components, we can draw out a component UML diagram. This is shown on the following picture:
- All the components shown on the previous diagram belongs to one global component - Button, but actions they perform are diferent. We must obtain these actions by programming them.





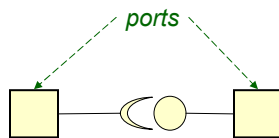
Ports

- Ports permit the interfaces of a component to be divided into discrete packets and used independently
- The externally visible behaviour of the component is the sum of its ports.



Συνδέοντας Εξαρτήματα Connecting Components

- Components can be connected by wiring together their **ports**
 - **connector**: a wire between two ports



connector by interfaces



delegation connector

(connect an external port with the port of a part component)

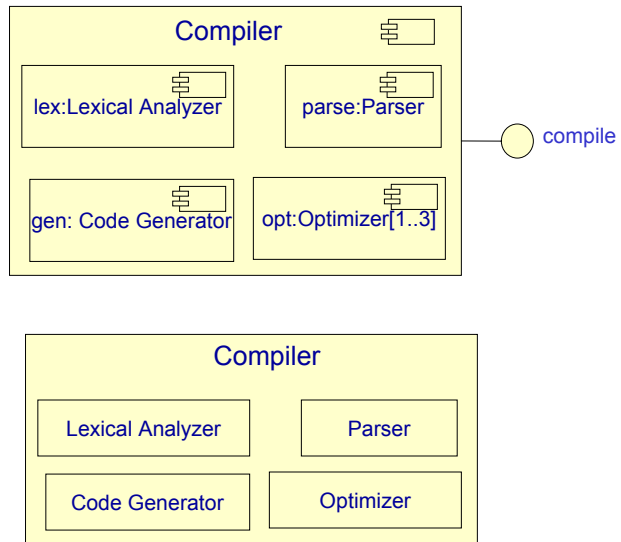


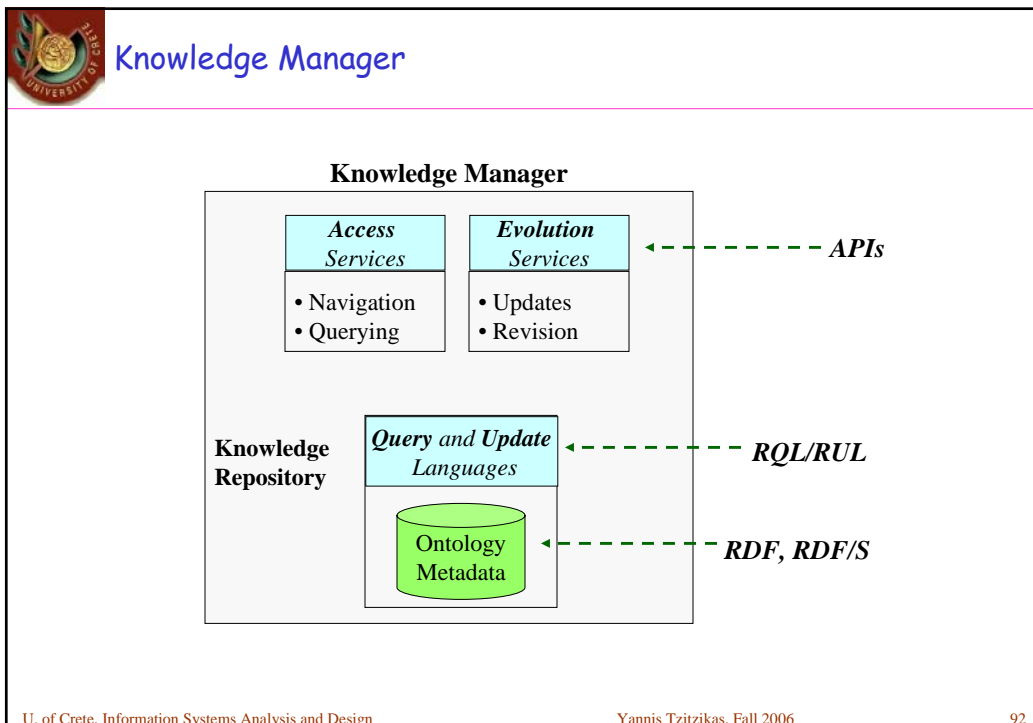
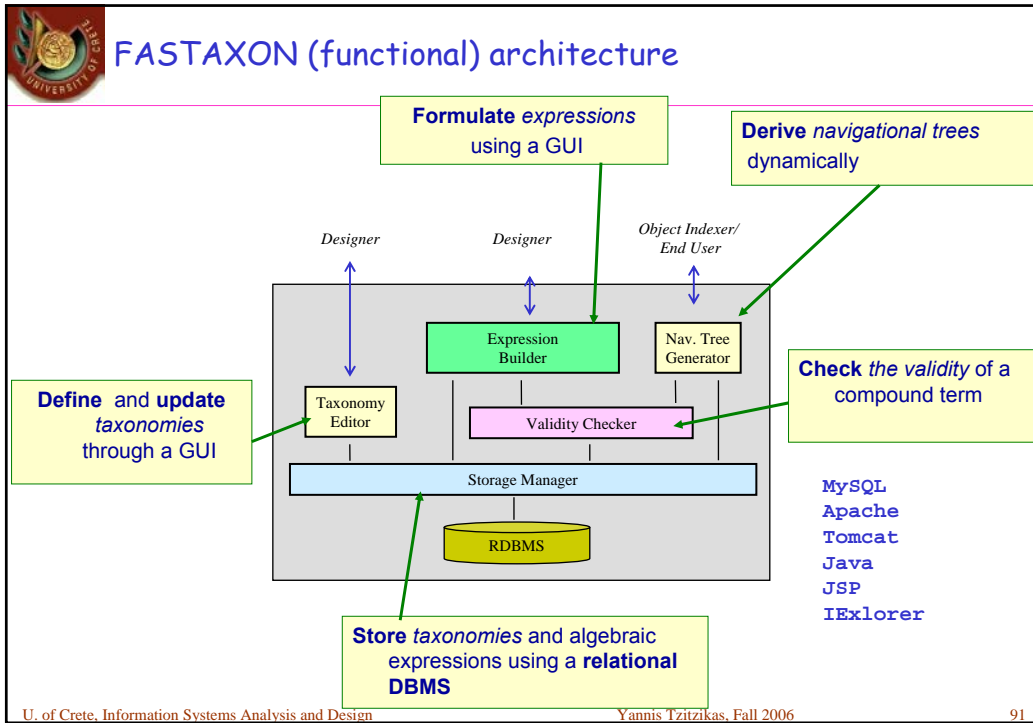
direct connector

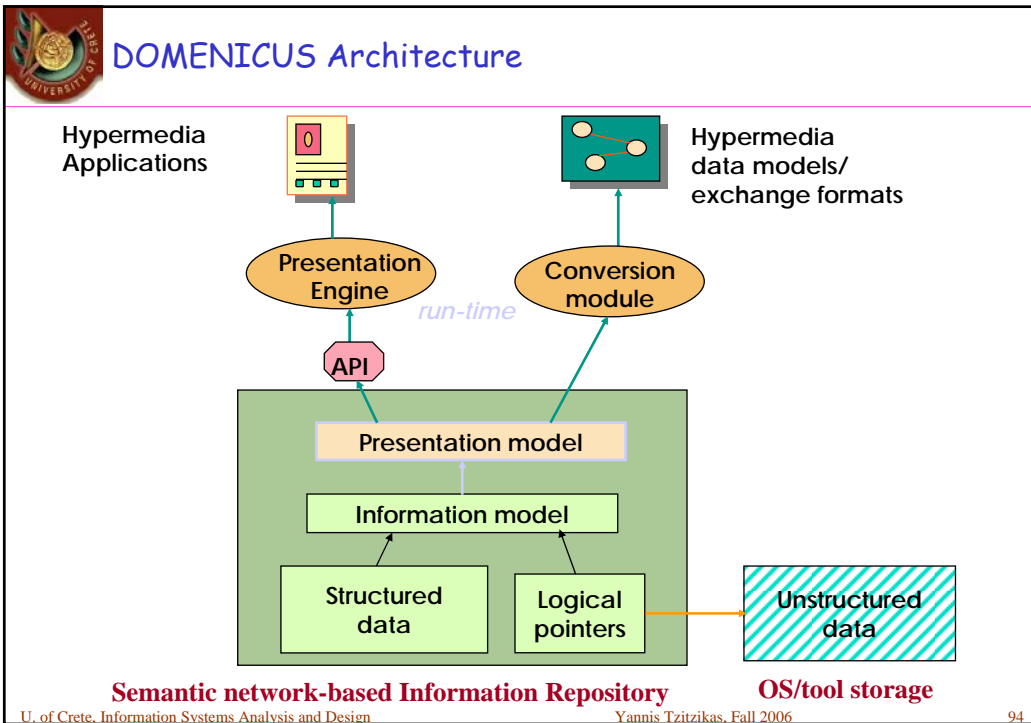
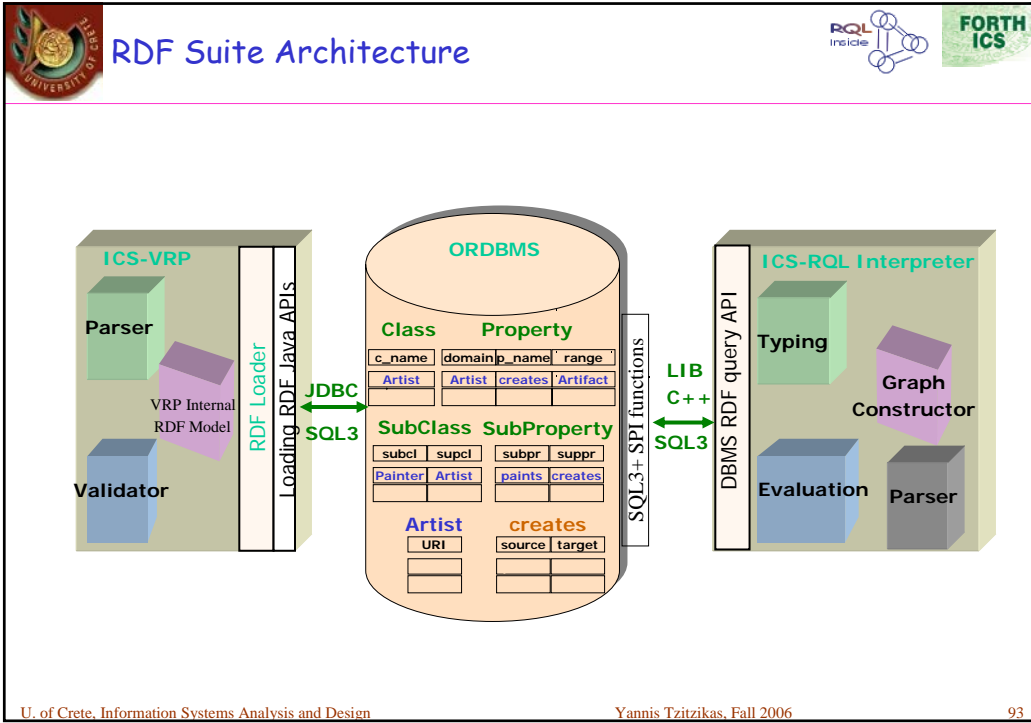
(more tight coupling)



In practice, components diagrams are sometimes depicted in a less formal and more liberal graphical notation









Διαγράμματα Παράταξης UML Deployment Diagrams

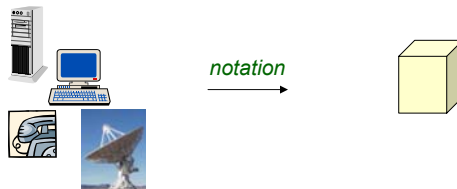


Deployment Diagrams (διαγράμματα ανάπτυξης/σύνταξης/παράθεσης)

Shows the physical relationship among software & hardware components in the delivered system

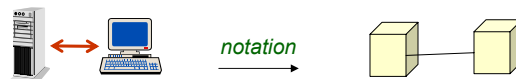
Node:

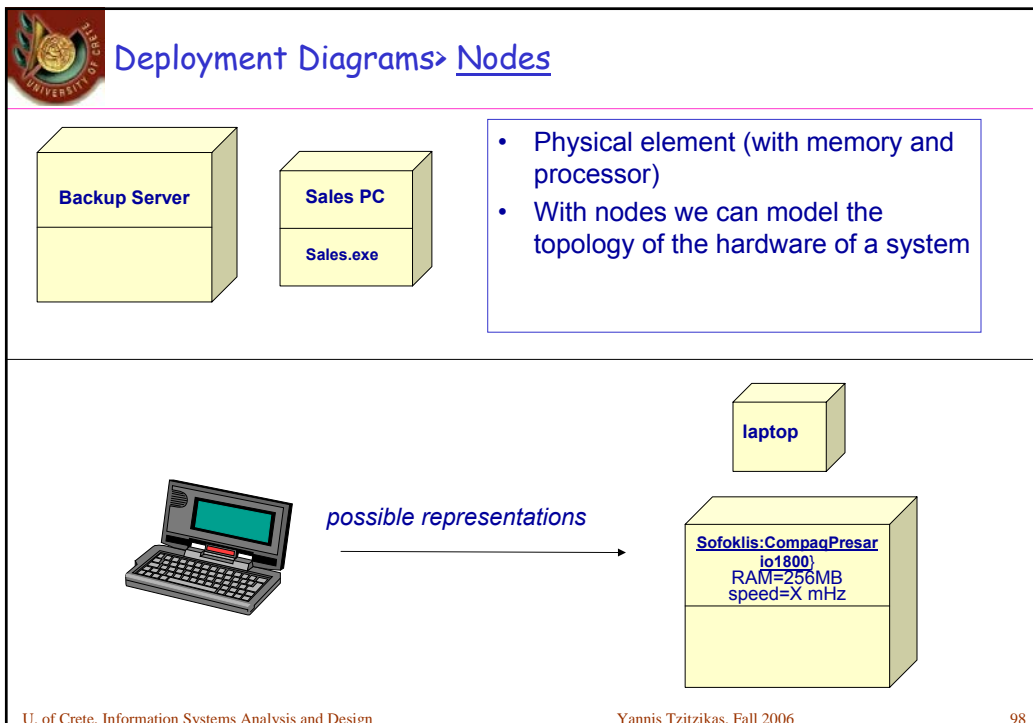
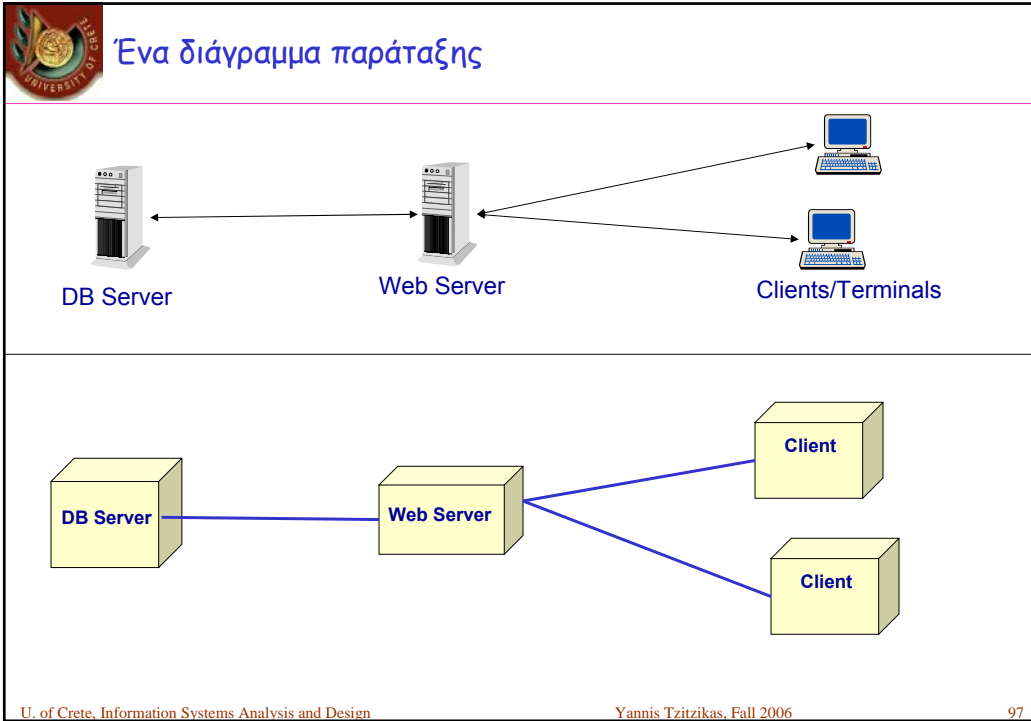
- computational unit (hardware)
 - e.g. PC, sensor, mainframe, mobile device



Connection (among nodes)

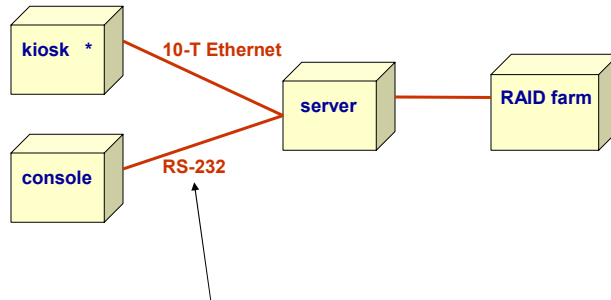
- communication paths over which the system will interact







Deployment Diagrams > Connections

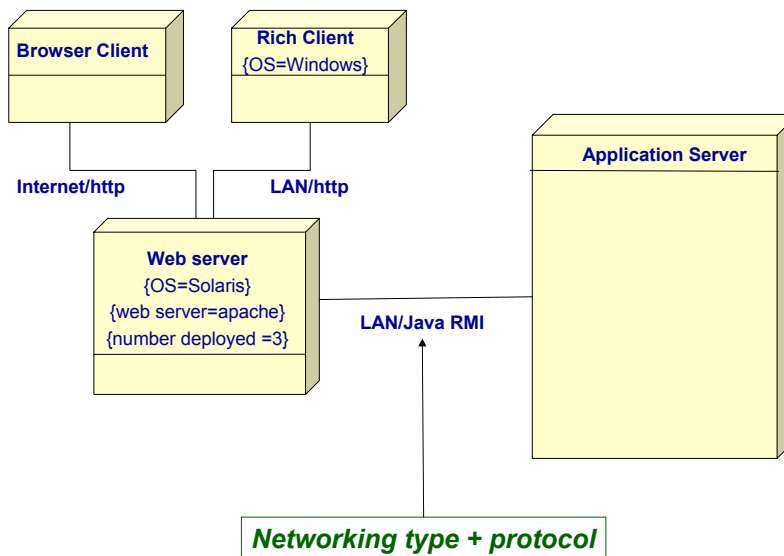


Connections

- Ethernet, serial line, satellite link
- we can use **stereotypes** to distinguish them to types
 - <<serial line>>
 - <<satellite link>>
 - ...

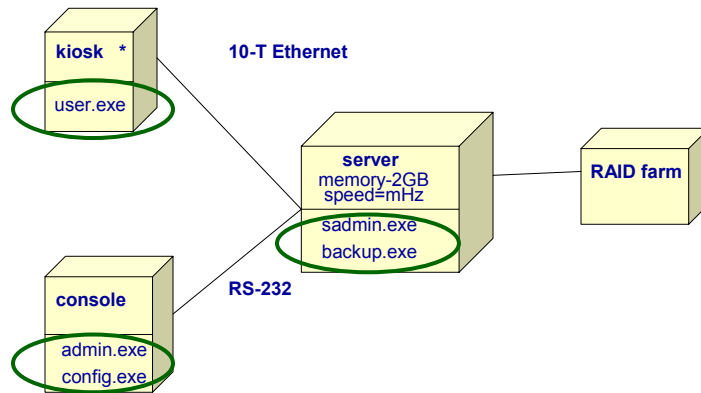


Deployment Diagrams > Connections





Παριστάνοντας την κατανομή των τεχνουργημάτων Modeling the Distribution of Artifacts

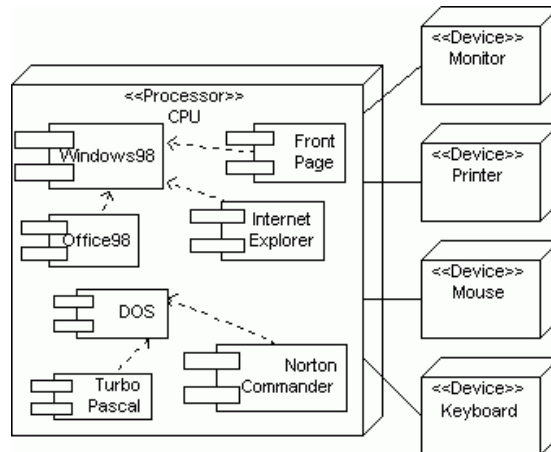


Συνδιάζοντας διαγράμματα Εξαρτημάτων και Παράταξης Combining Component and Deployment Diagrams



Παράδειγμα

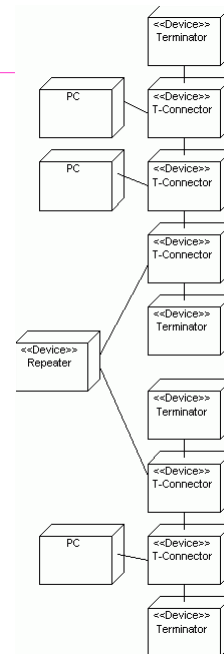
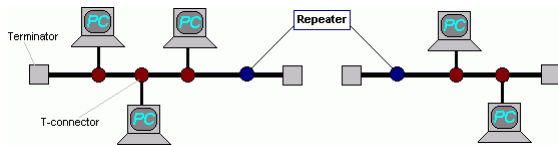
πηγή <http://odl-skopje.etf.ukim.edu.mk/uml-help/>



Another example:

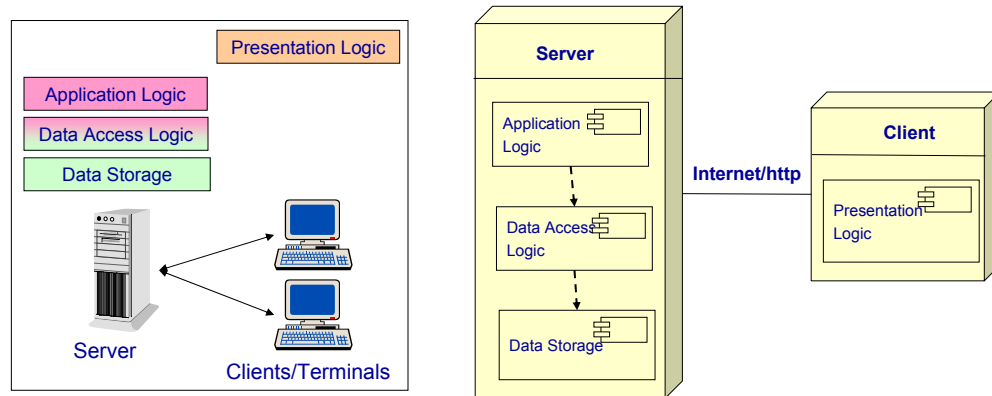
<http://odl-skopje.etf.ukim.edu.mk/uml-help/>

- Deployment diagram for ETHERNET





Combining Component and Deployment Diagrams: Example



Combining Component and Deployment Diagrams: Notes

- If we try to show all the components of a system in deployment diagrams they will probably become very large and difficult to read.
- So we usually depict the key elements
- Alternatively, (in case we want to show everything) we can use a table to denote artifacts and their locations (e.g. use Excel)



Hardware and Software Specification

- We have to specify the new hardware or software that must be purchased
- Actual acquisition of hardware and software usually left to a purchasing department -- especially in larger firms

Realities in Infrastructure Design

- Most often the infrastructure will be already in place
- Coordination of infrastructure components is very complex
 - The application developer will need to coordinate with infrastructure specialists

Steps in Hardware and Software Specification

- Note hardware in low-level network model to create list of needed hardware
- Describe equipment in as much detail as possible
- Consider whether increased processing and traffic will absorb unused hardware capacity
- Note all software running on each hardware component



Hardware

- **Commercial/Business**
 - Mainframes, Commercial Minicomputers, Microcomputers (Wintel: Windows on Intel), Embedded Systems
- **Technical/Engineering**
 - Supercomputers, Workstations and Servers (Sun SPARC), Microcomputers, Embedded Systems

Some distinctions:

- **Open vs Proprietary**
 - Proprietary: available by only one vendor (higher prices, low interoperability)
 - Open: available from many vendors (better prices, better interoperability)
- **Black-Box vs Glass-Box**
 - Black- box: only the vendor has access to its internals (e.g. bank ATM)
 - Glass Box: internals are accessible by the user, may replaceable by other vendor
 - Free UNIX derivatives (Linux, BSD) on Intel x86 with source code are glass-box systems



Δικτύωση Networking

- **Local Area Network**
 - short-distance (one building)
- **Backbone**
 - medium distance (campus)
- **Wide Area Network**
 - long-distance
- **Remote Access**
 - via phone / cable TV/satellite



Δικτύωση Networking

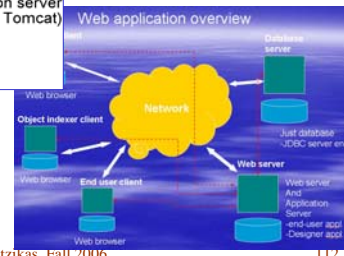
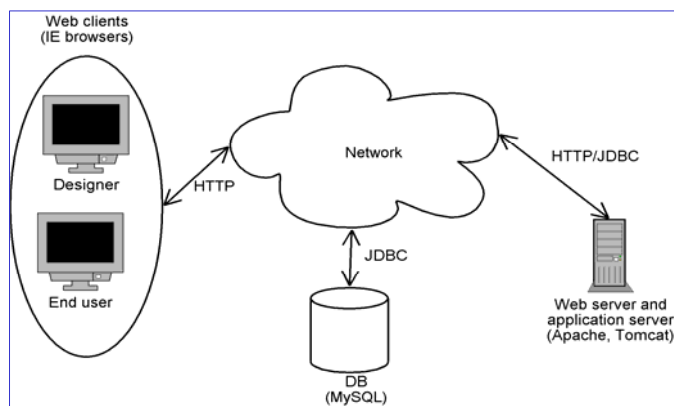
LAN	Backbone Network	WAN
<ul style="list-style-type: none"> • Ethernet <ul style="list-style-type: none"> – 10/100 Mb (1Gb fibre) – Inexpensive, widely used • Token Ring <ul style="list-style-type: none"> – 4/16 Mb – Not often used • ATM (copper) <ul style="list-style-type: none"> – 155 Mb (622Mb fibre) – Expensive, complex, flexible, high-overhead 	<ul style="list-style-type: none"> • 100 Mb (fibre) or Gb Ethernet <ul style="list-style-type: none"> – fast, inexpensive, simple • FDDI <ul style="list-style-type: none"> – Old 100 Mbit (increasingly obsolete) • ATM <ul style="list-style-type: none"> – 155 Mb, 622 MB 	<ul style="list-style-type: none"> • Long-distance line leased from telephone companies • Satellite links sometimes used
Remote Access	<ul style="list-style-type: none"> • Accessing a LAN or internet via phone/cable TV service <ul style="list-style-type: none"> – work from home, access when travelling, home internet service – Usually PPP over modem or cable modem • DSL services 	



Deployment diagrams are usually depicted
in a less formal and more liberal /vivid graphical notation

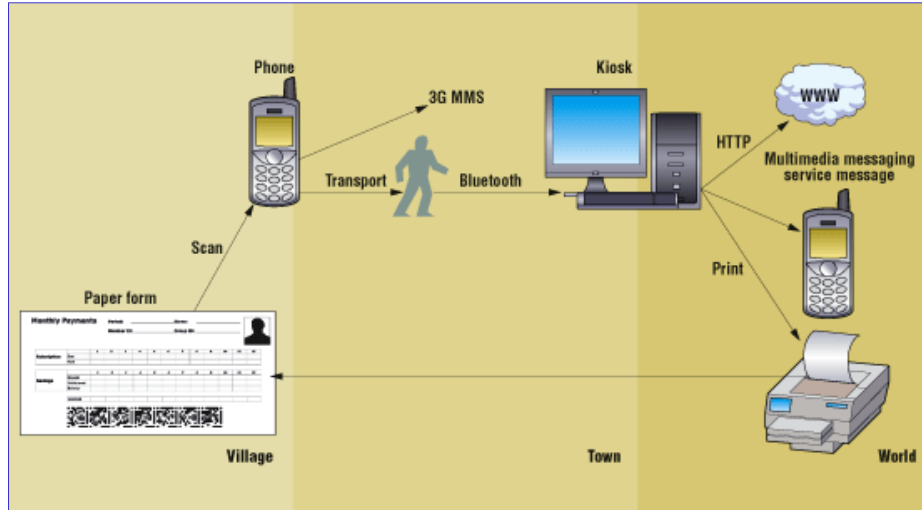


Deployment Diagrams: Examples (Fastaxon)

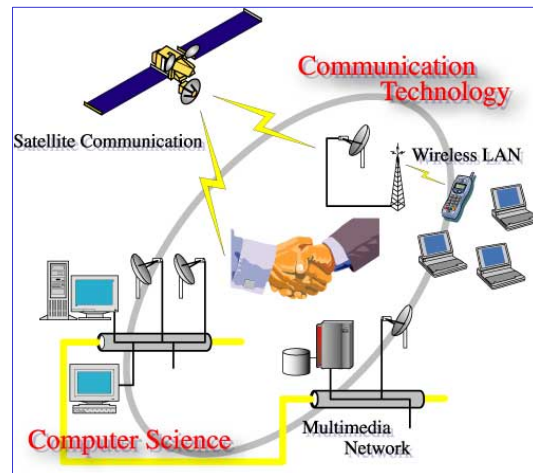




Deployment Diagrams: Examples

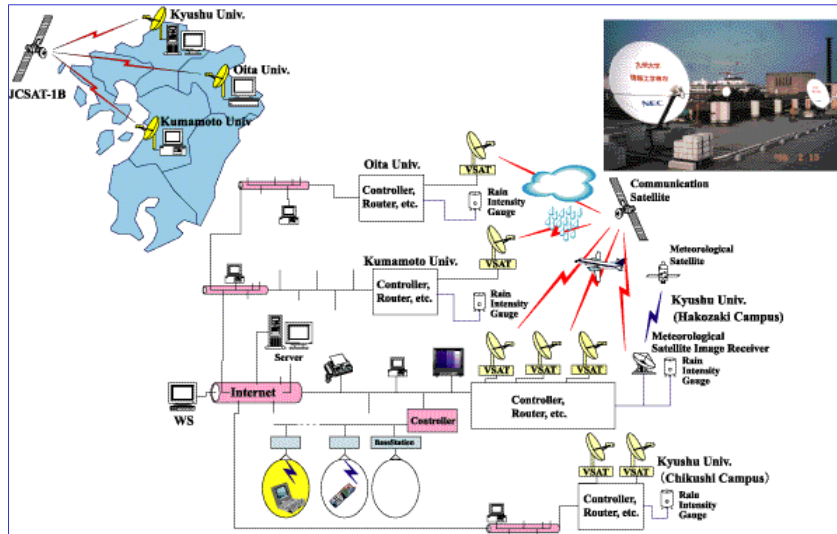


Deployment Diagrams: Examples

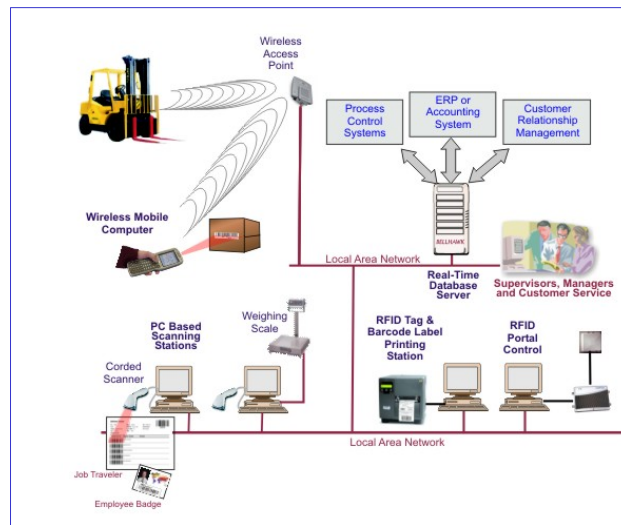




Deployment Diagrams: Examples

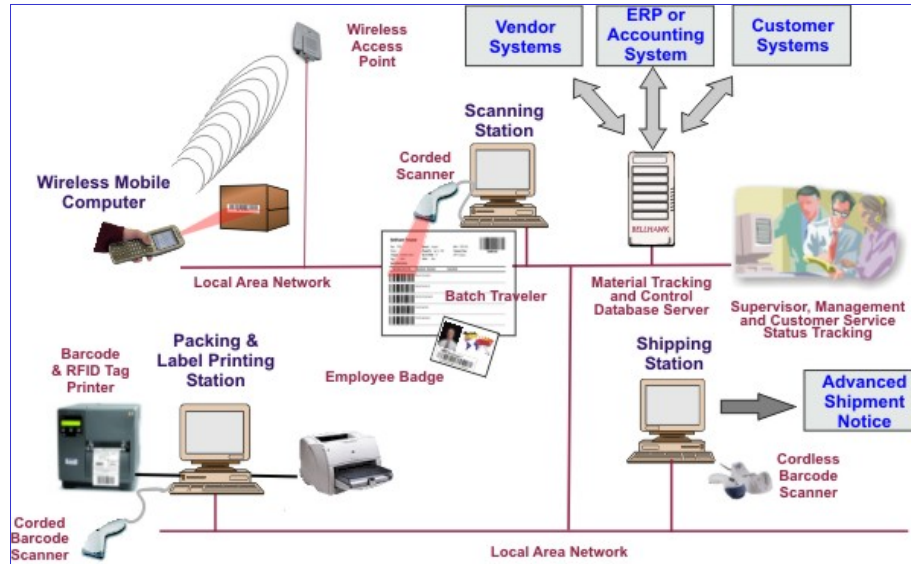


Deployment Diagrams: Examples





Deployment Diagrams: Examples



Deployment: Reading and References

- **UML Distilled: A Brief Guide to the Standard Object Modeling Language** (3rd Edition) by Martin Fowler, Addison Wesley, 2004. Chapter 8, Chapter 14 ([2nd Edition: Chapter 10](#))
- **The Unified Modeling Language User Guide** (2nd edition) by G. Booch, J. Rumbaugh, I. Jacobson, Addison Wesley, 2004 **Chapter 27**
- **Requirements Analysis and System Design** (2nd edition) by Leszek A. Maciaszek, Addison Wesley, 2005, Chapter 6
- **Object-Oriented Systems Analysis and Design Using UML** (2nd edition) by S. Bennett, S. McRobb, R. Farmer, McGraw Hill, 2002, **Chapter 19**
- <http://www.agilemodeling.com/artifacts/componentDiagram.htm>