SWINBURNE UNIVERSITY OF TECHNOLOGY

SWE30003
Software Architectures and Design

Lecture 8
Software Architectures and Architectural Styles

1

Logistical matters

- Weekly submissions – A & Q
  - Week 2: 214 and 202 out of 270;
  - Week 3: 222 and 215 out of 270;
  - Week 4: 202 and 199 out of 266;
  - Week 5: 217 and 210 out of 265
  - Week 6: 213 and 211 out of 265;
  - Week 7: 194 and 186 out of 265;
  - Week 8: (xx students not meeting 40% req to date, 3 sets left);
  - **No late submission, hurdle requirement**
- Assignment 2: deadline getting close …

2

2

Question to Answer – week 7

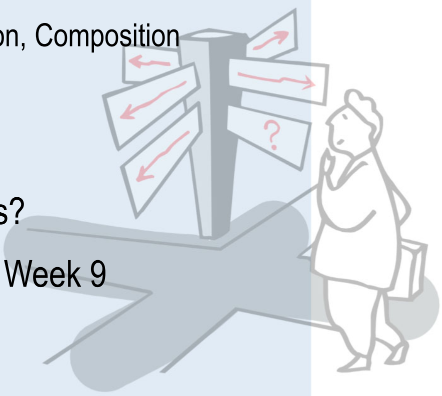
What is the difference between roles and classes in a Design Pattern? Are they the same thing? Pick two design patterns and identify the different classes and roles.

3

3

Outline

- **Software Architectures**
  - Processing, Interaction, Composition
- Architectural Styles
  - Deployment
- Why use Architectures?
- Required Reading for Week 9



4

4

Principal References

- Ian Sommerville, *Software Engineering* (8th Edition), Addison-Wesley, 2007, Chapters 11 to 13.
- *Desmond F. D'Souza and Alan Cameron Wills, Objects, Components and Frameworks with UML*, Addison-Wesley, 1999, Chapter 12.
- Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal, *Pattern-Oriented Software Architecture: A System of Patterns*, Addison-Wesley, 1996, Chapters 2 and 6.
- Mary Shaw and David Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996, Chapter 2.

5

5

Recap - Software Abstractions

- *Software Architectures*
- Subsystems
- Components
- Modules, Packages
- Objects, Interfaces
- Abstract Data Types
- Functional abstractions (i.e. procedures)
- Primitive data types
- Simple arithmetic expressions and control structures
- Macros
- Symbolic names (for instructions and memory cells)
- Machine instructions, memory cells

"high-level"

"mid-level"

"low-level"

6

6

Subsystems, Module, Components

- A *sub-system* is a system in its own right whose operation is independent of the services provided by other sub-systems.
- A *module* is a system component that provides services to other components but would not normally be considered as a separate system.
- A *component* is an independently deliverable unit of software that encapsulates its design and implementation and offers interfaces to the out-side, by which it may be composed with other components to form a larger whole.

☞ *Terms are often used inconsistently! We may use the term "**processing element**" instead.*

7

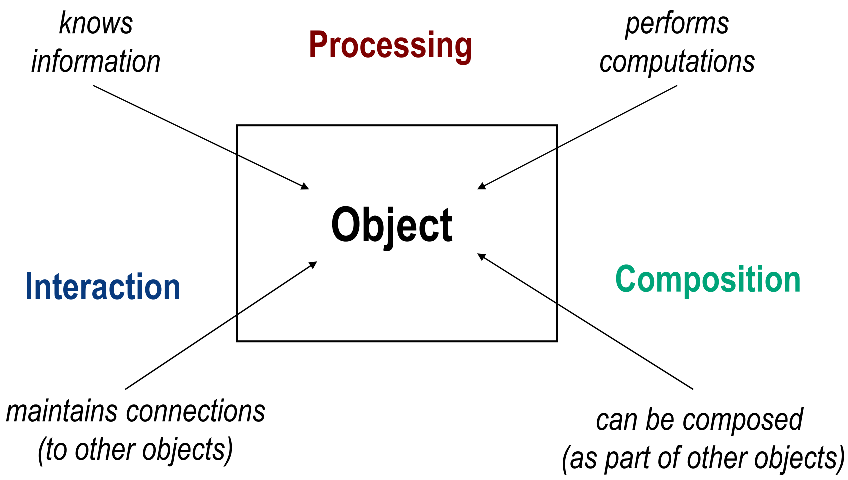
7

Common to all contemporary software design methods: support for *abstraction*, *decomposition*, and *composition*!

8

8

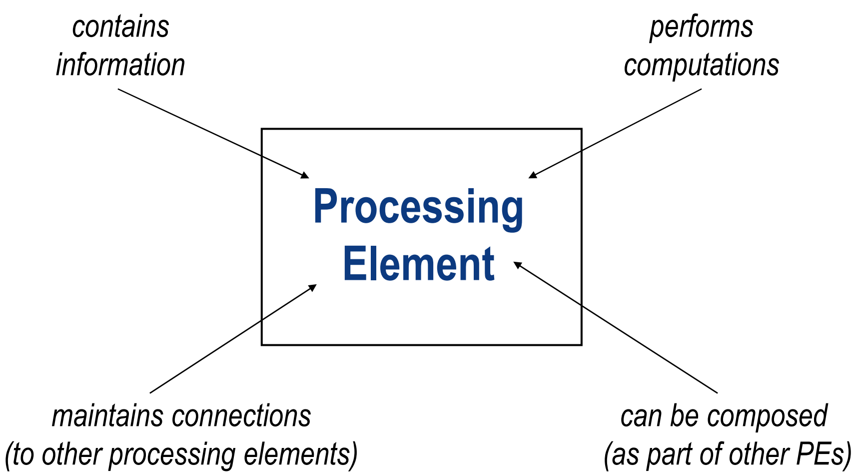
Recap – what is an Object



9

9

In fact…



10

10

Software Architecture – one view

"In most successful software projects, the expert developers working on that project have a shared understanding of the system to be implemented. This shared understanding is often called 'architecture'.

It includes how the system is divided into processing elements (i.e. components) and their interaction through interfaces. Theses components are usually composed of smaller components, but the architecture only includes the ones that are understood by all developers."

— Ralph Johnson, 2003

11

11

Example – Enterprise Systems

Presentation Layer

- Exposes data and behaviour to users
- Support for various user tasks
- Typically event-based programming style used

↕

Domain Layer

- Defines the "business logic" of system (includes business rules, business goals, workflows etc.)
- No particular programming style required (but often OO is used)

↕

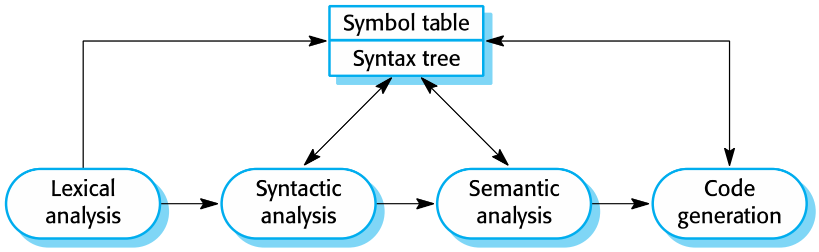
Data Source Layer

- Specifies the data of the system
- Supports CRUD (creating, reading, updating, deleting of data)
- Declarative Data Description and Query Language (e.g., SQL)

12

12

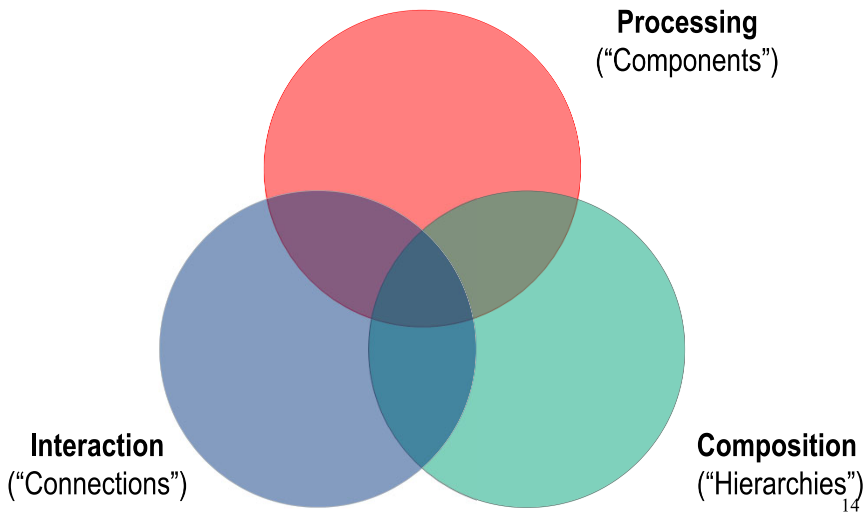
Example – Compiler



13

13

Main Focus – Separation of Concerns



14

14

Software Architecture (cont.)

The architecture of a software system is described as:

- the *structure(s) of its high-level processing elements*,
- the *externally visible properties* of the processing elements, and
- the *relationships and constraints* between them.

in other words:

The set of *design decisions* about any system (or subsystem) that keeps its implementors and maintainers from exercising *"needless creativity"*.

15

15

Software Architecture (cont.)

"A software architecture is a set of architectural (design) elements that have a particular form. Properties constrain the choice of architectural elements whereas rationale captures the motivation for the choice of elements and form."

— Dewayne Perry and Alexander Wolf, 1992

16

16

Rationale

Rationale explains why a decision was made and what the implications are in changing it! Use rationale to explain:

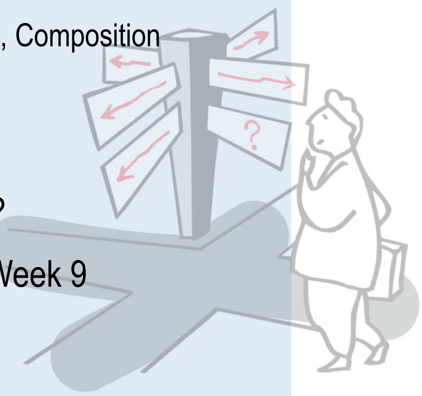
- implications of *system-wide* design choices on meeting requirements,
- effects on the architecture in the context of changing/adding requirements,
- design alternatives that were rejected (and why!),
- …

17

17

Outline

- Software Architectures
  - Processing, Interaction, Composition
- **Architectural Styles**
  - Deployment
- Why use Architectures?
- Required Reading for Week 9



18

18

Moscow State University

19

19

Palace of Culture, Warsaw

20

20

Architectural Styles

"An architectural style defines a family of software systems in terms of their structural organization. An architectural style expresses components and the relationships between them, with the constraints of their application, and the associated composition and design rules for their construction."

— Dewayne Perry and Alexander Wolf, 1992

21

21

Popular Architectural Styles

- Data-flow architectures:
  - Batch sequential, Pipes-and-Filter
- Call-and-Return architectures:
  - Client-Server
  - Layered
  - Tiered architectures
  - Object-Oriented
  - Main program and subroutine
- Data-centred architectures:
  - Repository, Blackboard
- Virtual machine architectures:
  - Interpreter, rule-based systems
- Independent Component Architectures:
  - Peer to Peer
  - Event-Driven

22

22

Layered Architectures

- A *layered architecture* organises a system/module/component into a set of layers each of which provide a set of services to the layer "above" and uses services from the layer(s) "below".
- Normally layers are constrained so elements only "see"
  - other elements in the same layer, or
  - elements of the layer below.
- Call-backs may be used to communicate to higher layers.
- Supports the incremental development of sub-systems in different layers:
  - When a layer interface changes, only the adjacent layer is affected.

23

23

Example – Enterprise System revisited

User interface

User interface management
Authentication and authorization

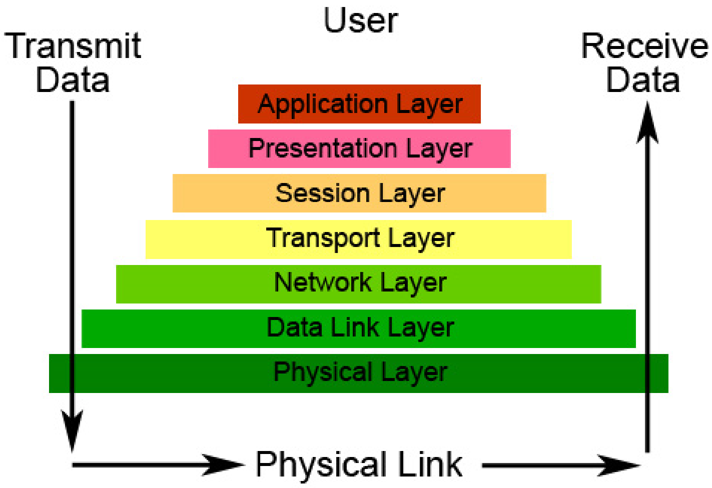
Core business logic/application functionality
System utilities

System support (OS, database etc.)

24

24

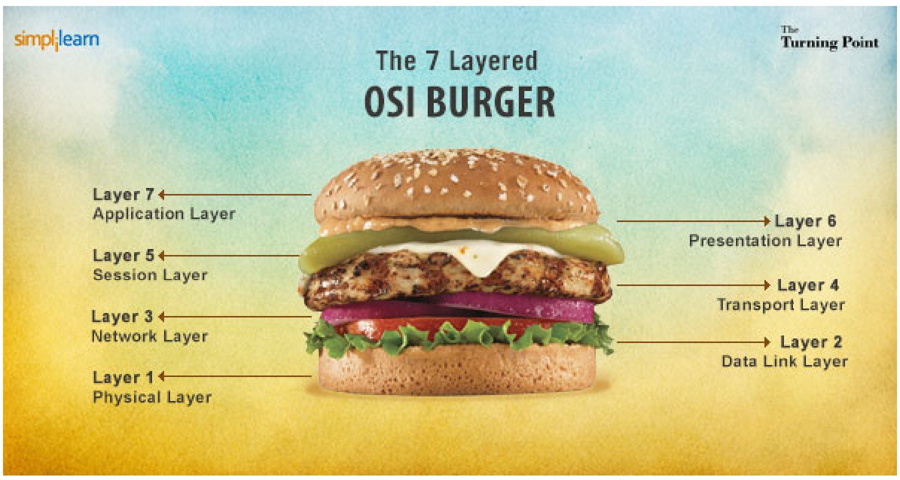
OSI Reference Model



25

25

OSI Reference Model (cont.)



26

26

Dataflow Architectures

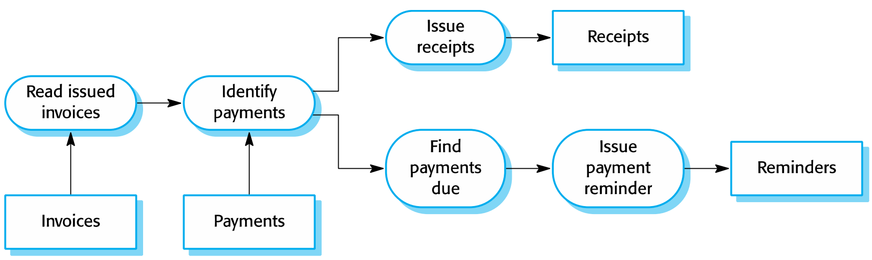
In a dataflow architecture each component performs *functional transformations* on its *inputs* to produce *outputs*.

- Main elements:
  - ☐ Data source(s) and Data sink(s)
  - ☐ Filters and Transformers
- Dataflow architectures are generally free of cycles (unless *feedback-loops* are introduced),
- Not really suitable for *interactive systems*

27

27

Example – Invoice Processing System



28

28

Pipes-and-Filters

- The processing of the data in a system is organized so that each processing element (filter) is discrete and carries out *one type* of data transformation.
- The data flows (as in a pipe) from one processing element to another for processing.
- In general, single input and single output.
- Example: Unix Bourne shell:

```
tar -cvf - . | gzip -9 | ssh mercury dd
```

29

29

Break

30

30

Repository Architectures

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

Advantages:

- Efficient way to share large amounts of data.
- Components need not be concerned with how data is produced, backed up etc.
- Sharing model is published as the repository schema.
- Easy to implement coordination schemes between components.

31

31

Repository Architectures (cont.)

Disadvantages:

- Components must agree on a repository data model.
- Data evolution is difficult and expensive.
- No scope for specific management policies.
- Difficult to distribute efficiently.

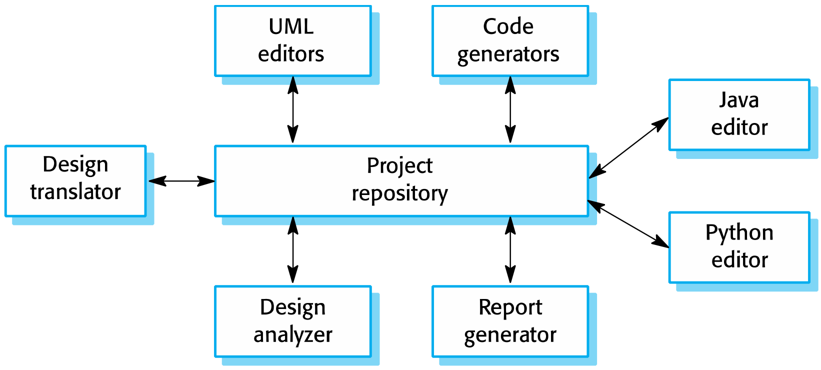
Examples:

- Database based information systems
- Document repositories (CVS, Subversion, Git)

32

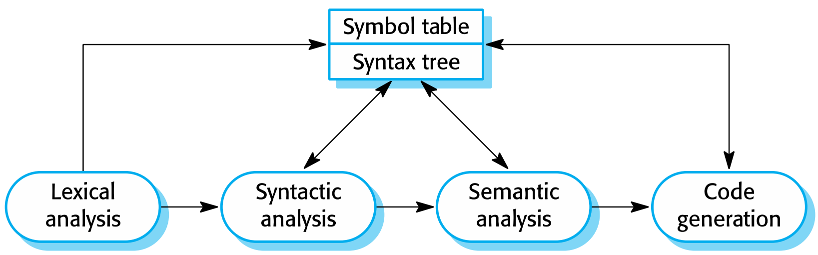
32

Example – Repository Architecture for IDE



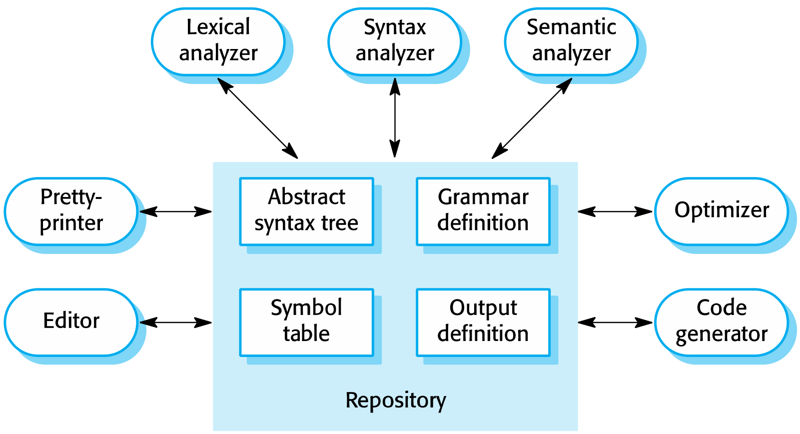
33

Example – Compiler



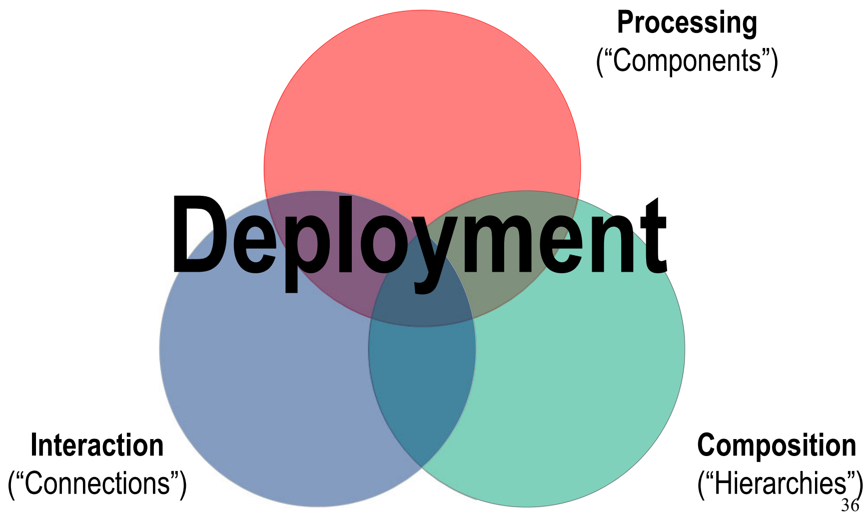
34

Repository Architecture for a Compiler

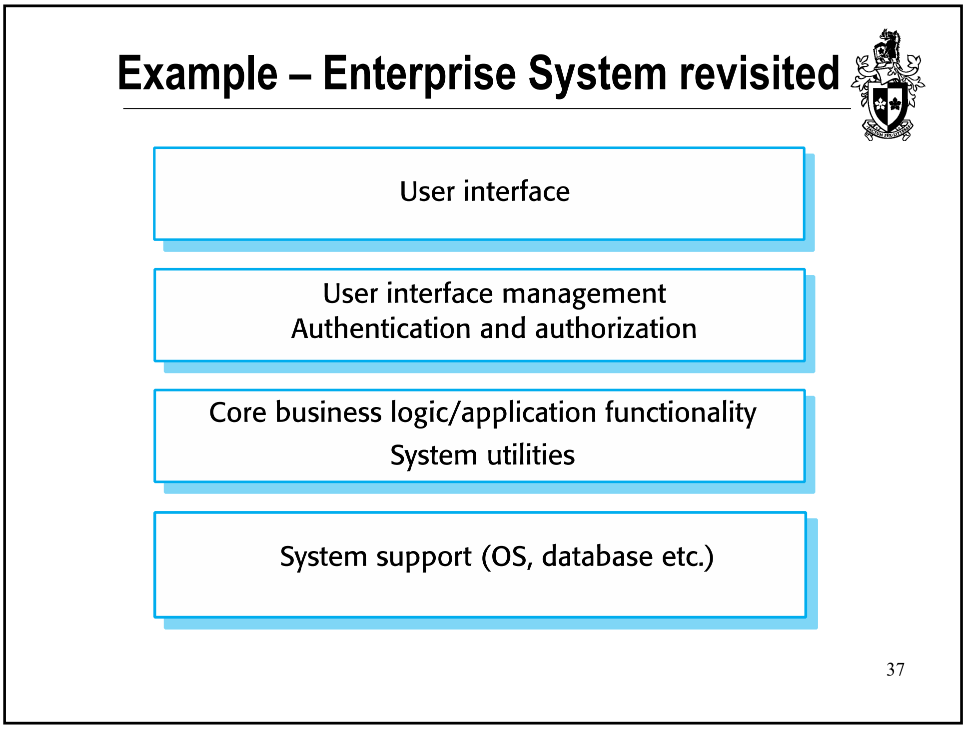


35

Separation of Concerns – Take 2

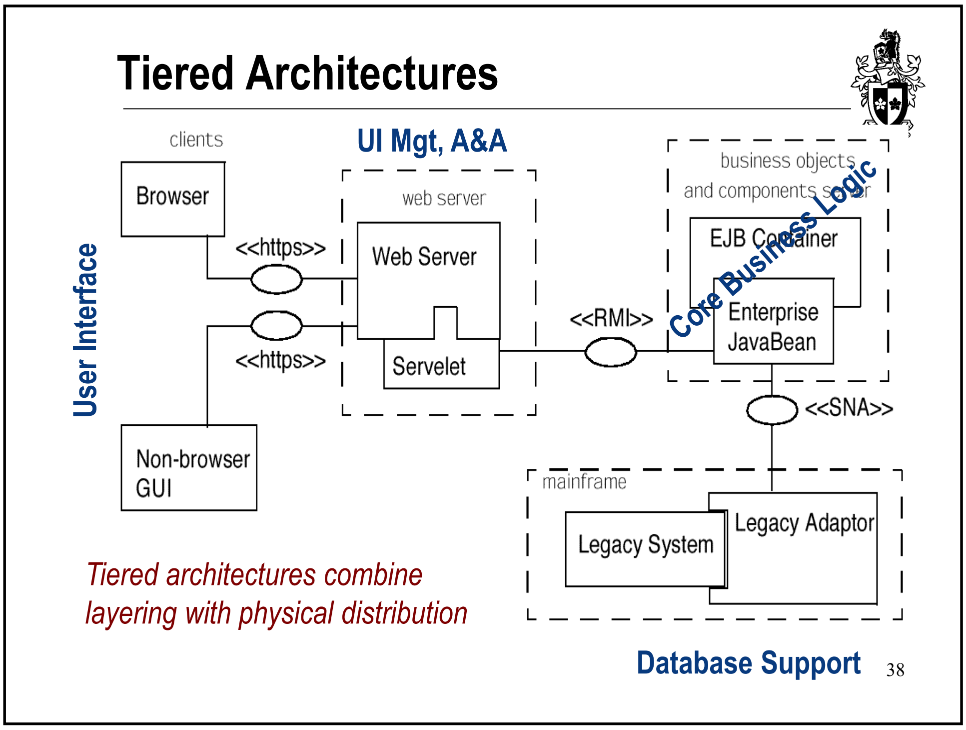


36



Example – Enterprise System revisited
User interface
User interface management
Authentication and authorization
Core business logic/application functionality
System utilities
System support (OS, database etc.)
37

37



Tiered Architectures
clients
UI Mgt, A&A
business objects
and components
Browser
web server
<<https>>
Web Server
EJB Container
Core Business Logic
User Interface
<<RMI>>
Enterprise
JavaBean
<<https>>
Servelet
<<SNA>>
Non-browser
GUI
mainframe
Legacy Adaptor
Legacy System
Tiered architectures combine
layering with physical distribution
Database Support
38

38

Popular Architectural Styles (cont.)

- Call-and-Return architectures:
  - Client-Server
- Independent component architectures:
  - Peer to Peer
  - Event systems: implicit invocation, explicit invocation

39

39

Client-Server Architectures

A client-server architecture *distributes application logic and services* respectively to a number of client and server sub-systems, each (potentially) running on a different machine and communicating through the network (e.g., by RPC).

40

40

Client-Server Architectures (cont.)

Advantages

- *Distribution* of data is straightforward
- Makes *effective use of networked systems*. May require cheaper hardware
- Easy to *add new servers* or upgrade existing servers

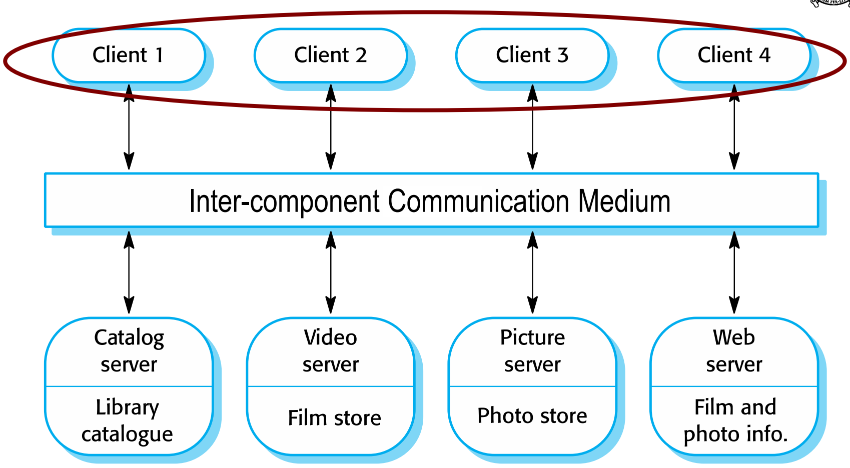
Disadvantages

- *No shared data model* so sub-systems use different data organization.
  Data interchange may be inefficient
- *Redundant management* in each server
- May require a *central registry* of names and services — it may be hard to find out what servers and services are available

41

41

Example – Film and Picture Library



42

42

Event-driven Architectures

In an event-driven architecture components perform services in *reaction to external events* generated by other components.

- In *interrupt-driven* models real-time interrupts are detected by an interrupt handler and passed to some other component for processing.
- In *broadcast* models an event is broadcast to all sub-systems. Any sub-system which can handle the event may do so.

43

43

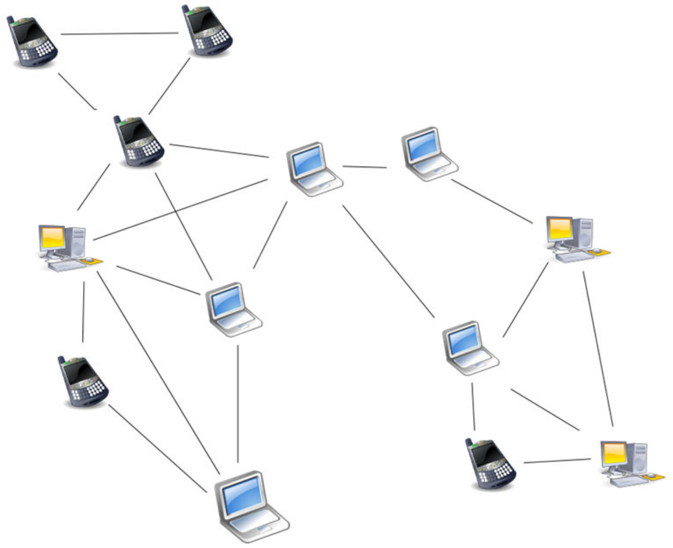
Peer-To-Peer Architectures

- A peer-to-peer (P2P) architecture is a type of *decentralized* and (often) distributed structure in which individual nodes ("peers") act both as *suppliers and consumers of resources*.
- In a peer-to-peer network, tasks (e.g., sharing audio/video) are shared amongst multiple (often loosely) interconnected peers who each make a portion of their resources available to other peers, *without centralized coordination*.
- Peers are not necessarily static, they can come/go "randomly"
- Examples:
  - BitTorrent
  - Bitcoin
  - Napster

44

44

Peer-To-Peer Architectures (cont.)



45

45

Heterogeneous Styles

Large software systems rarely conform to a single architectural model (or style). They incorporate *different styles at different levels of abstraction*.

- *locationally* heterogeneous: the architectural structure of a system reveals different styles in different areas
  - branches of a main-program-and-subroutines system have a shared data repository.
- *hierarchically* heterogeneous: a component of one style, when decomposed, is structured according to the rules of a different style
  - an element of a pipe and filter pipeline is structured in a layered style.
- *simultaneously* heterogeneous: a system may be described using several architectural styles
  - a complier can be viewed both as a dataflow and/or repository system.

46

46

Reference Models and Architectures

- A reference model is a division of functionality together with data flow between the resulting elements
  - can be considered as a *standard decomposition of a know problem* into parts that cooperatively solve the problem.
- A reference architecture is a reference model mapped onto software components and the data flow(s) between these components
  - i.e. a mapping of system functionality onto a system decomposition.

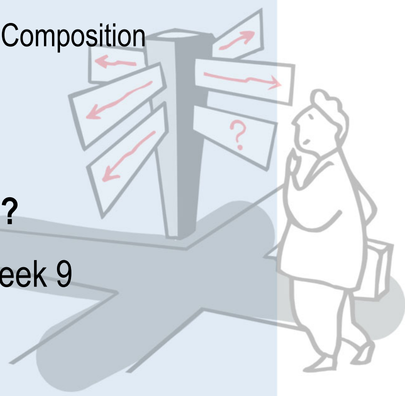
☞ *A reference architecture can also be viewed as a combination of a reference model and an architectural style.*

47

47

Outline

- Software Architectures
  - Processing, Interaction, Composition
- Architectural Styles
  - Deployment
- **Why use Architectures?**
- Required Reading for Week 9



48

48

Conway's Law

Large software projects will generally divide into multiple groups — the resulting product will always reflect the structure of these groups (and the way they communicate!):

> *"Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations."*
>
> — Melvin Conway, 1968

Example:
"If you have four groups working on a compiler, you will get a 4-pass compiler."

49

49

Architectural Parallels

- Architects are the *technical interface* between the customer and the contractor building a system.
- A bad architectural design for a building *cannot be rescued by good construction* — the same is true for software.
- There are *specialized types* of building architects and software architects.
- There are *schools or styles* of building architecture and software architecture.

50

50

Architectures and Design

"The use of architectures and architectural styles is a different approach to develop software. [...] It is often not desirable to design a system from scratch, but to look for already existing software systems that have solved a similar problem or a problem in the same application domain. Reusing the architecture of such systems has the advantage that the new system benefits from well-understood properties and important design decisions of existing systems."

51

51

Why use Architectures?

- Architectural styles document existing, well proven design experience.
- Architectures identify and classify abstractions that are at a higher level of abstraction than simple programming language constructs (and often design patterns).
- Architectural styles provide a *common vocabulary* and understanding for design principles.
- Reference architectures support the construction of software with well-defined properties.
- Guidelines help to choose suitable architecture(s) given the problem domain and/or required quality attributes.

☞ *Architectures help to manage software complexity.*

52

52

Additional Recommended Readings

- Martin Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2002
- David M. Dikel, David Kane and James R. Wilson, *Software Architecture: Organizational Principles and Patterns*, Prentice Hall, 2001
- Paul C. Clements, Rick Kazman and Mark Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison-Wesley, 2002
- Paul Clements, et.al. *Documenting Software Architectures: Views and Beyond*, Addison Wesley, 2002
- Software Architecture Resources http://www.serc.nl/people/florijn/interests/arch.html

53

53

Questions for Review

1. Can one determine quality attributes at an architecture level without waiting until it has been developed and deployed?
2. What is the difference between Software Design Patterns and Architectural Patterns? Is one type more important than the other?
3. Could software architecture be used to analyse problems or just a representation of the current system?
4. What (if any) information should be included in a software architecture design which could never be mechanically reverse engineered from the undocumented source code of an existing system?
5. Give a brief description of what Software Architecture means to you. Give an example of why Software Architecture is important.

54

54

More Questions for Review…

- What is meant by a “fat client” or a “thin client” in a 4-tier architecture?
- What kind of architectural styles are supported by the Java AWT? by RMI?
- How do callbacks reduce coupling between software layers?
- How would you implement a dataflow architecture in Java?
- Is it easier to understand a dataflow architecture or an event-driven one?
- What are the coupling and cohesion characteristics of each architectural style?

55

55

Question to Answer – week 8 (for week 9)

The spec of the “Question to Answer” is under the corresponding assignment setup, which will be released after this lecture.

56

56

Exercise – Keywords in Context

"The Key Word in Context (KWIC) index system accepts an ordered set of lines, each line is an ordered set of words, and each word is an ordered set of characters. Any line may be "circularly shifted" be repeatedly removing the first word and appending it at the end of the line. The KWIC index system outputs a listing of all circular shifts of all lines in alphabetical order [ignoring case]."

David Parnas, *On the Criteria to be Used in Decomposing Systems into Modules*, 1972

57

57

Required Reading Week 9

- Len Bass, Paul Clements, and Rick Kazman, *Software Architecture in Practice* (3rd Edition), Addison-Wesley, 2013, Chapter 13. (available online through Swinburne Library
- Len Bass, Paul Clements, and Rick Kazman, *Software Architecture in Practice* (1st Edition), Addison-Wesley, 1998, Chapter 5 (available electronically from Canvas).

58

58