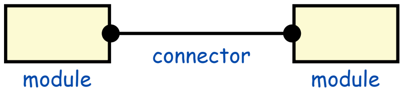
Lecture 4: Showing the architecture

- → Coupling and Cohesion
- → UML Package Diagrams
- → Software Architectural Styles:
  - Layered Architectures
  - Pipe-and-filter
  - Object Oriented Architecture
  - Implicit Invocation
  - Repositories

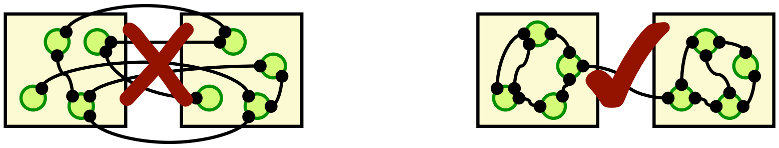
Coupling and Cohesion

Architectural Building blocks:



A good architecture:

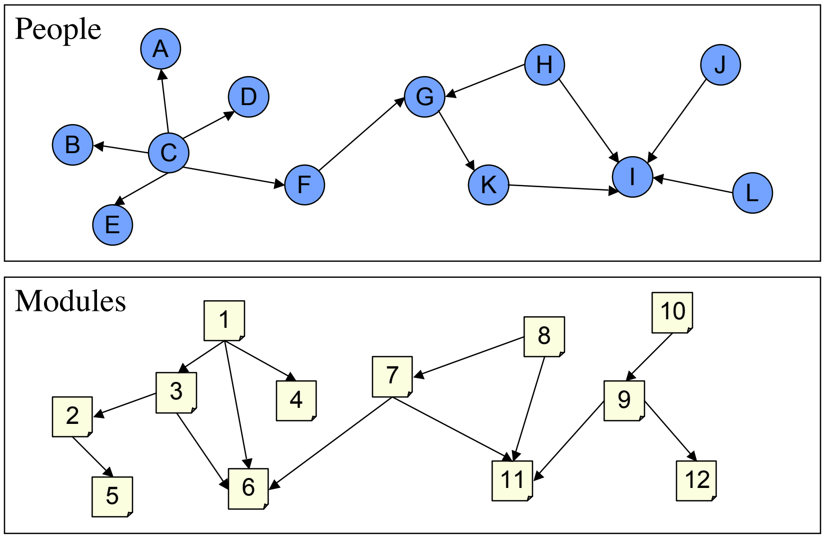
- Minimizes coupling between modules:
  - Goal: modules don't need to know much about one another to interact
  - Low coupling makes future change easier
- Maximizes the cohesion of each module
  - Goal: the contents of each module are strongly inter-related
  - High cohesion makes a module easier to understand



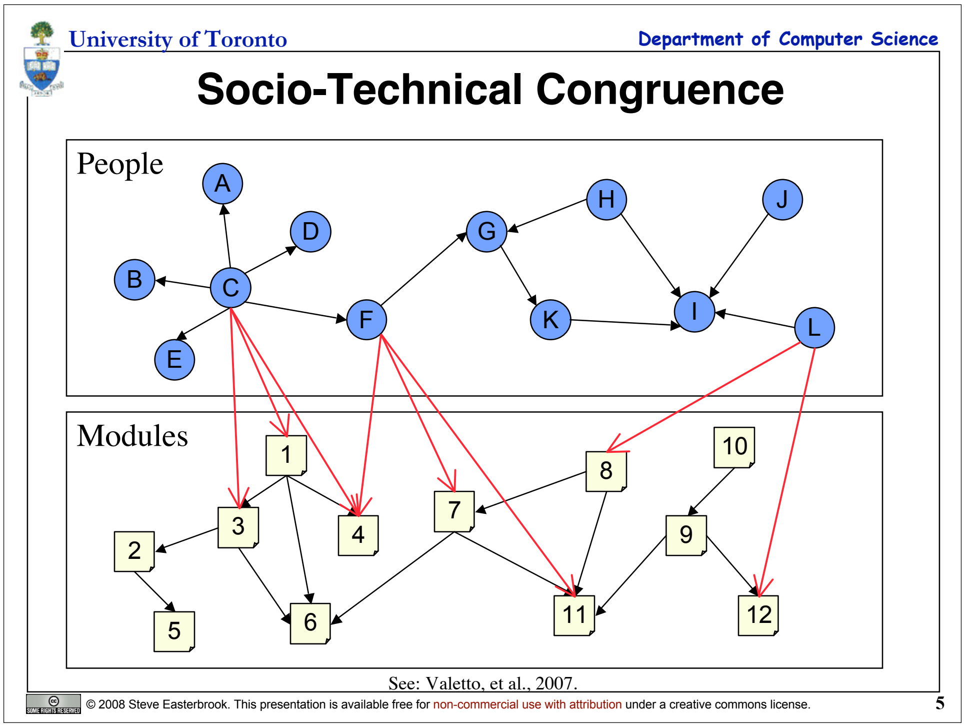
Conway's Law

"The structure of a software system
reflects the structure of the organisation
that built it"

Socio-Technical Congruence

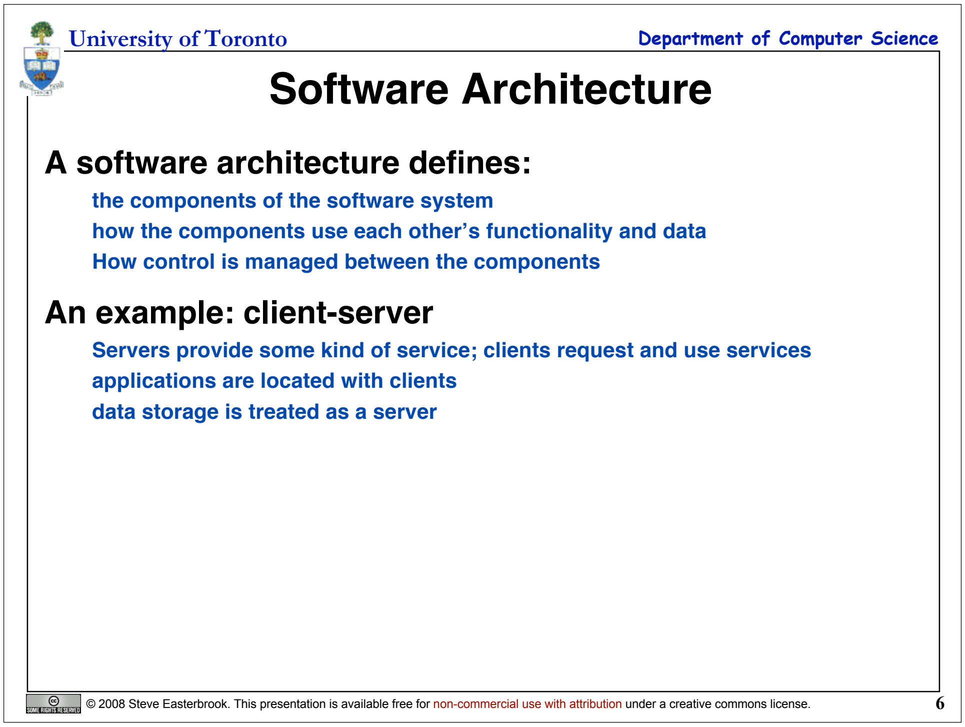


See: Valetto, et al., 2007.



Socio-Technical Congruence

See: Valetto, et al., 2007.



Software Architecture

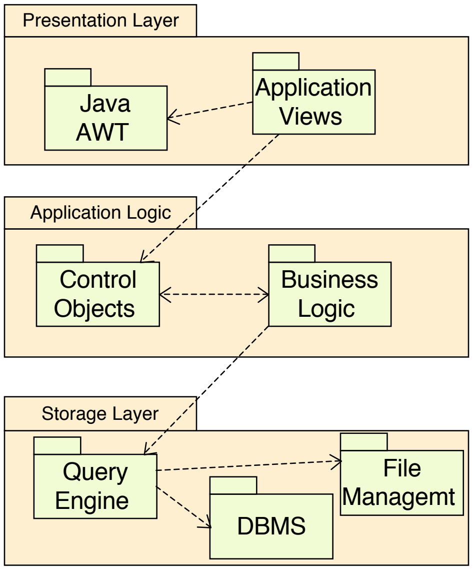
A software architecture defines:

- the components of the software system
- how the components use each other's functionality and data
- How control is managed between the components

An example: client-server

- Servers provide some kind of service; clients request and use services
- applications are located with clients
- data storage is treated as a server

3-layer architecture



UML Packages

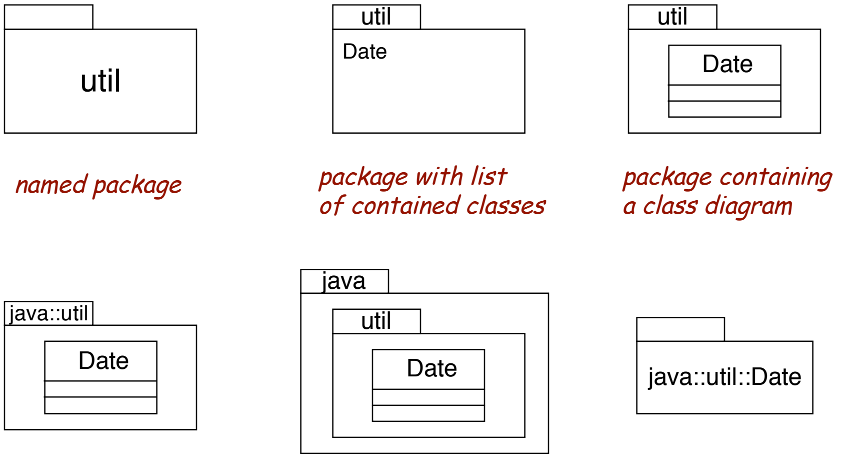
We need to represent our architectures

- UML elements can be grouped together in packages
- Elements of a package may be:
  - ➢ other packages (representing subsystems or modules);
  - ➢ classes;
  - ➢ models (e.g. use case models, interaction diagrams, statechart diagrams, etc)
- Each element of a UML model is owned by a single package

Criteria for decomposing a system into packages:

- Ownership
  - who is responsible for working on which diagrams
- Application
  - each problem has its own obvious partitions;
- Clusters of classes with strong cohesion
  - e.g., course, course description, instructor, student,…
- Or use an architectural pattern to help find a suitable decomposition

Package notation



named package

package with list of contained classes

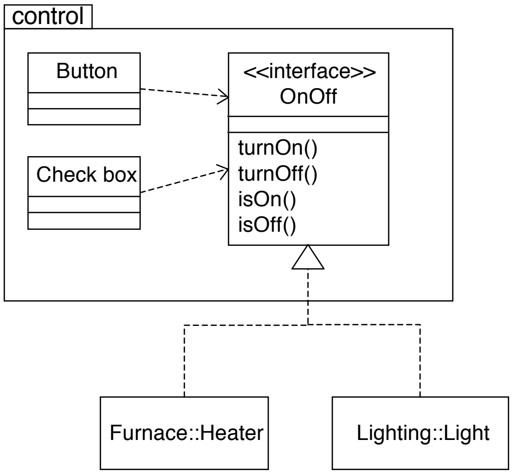
package containing a class diagram

package with qualified name

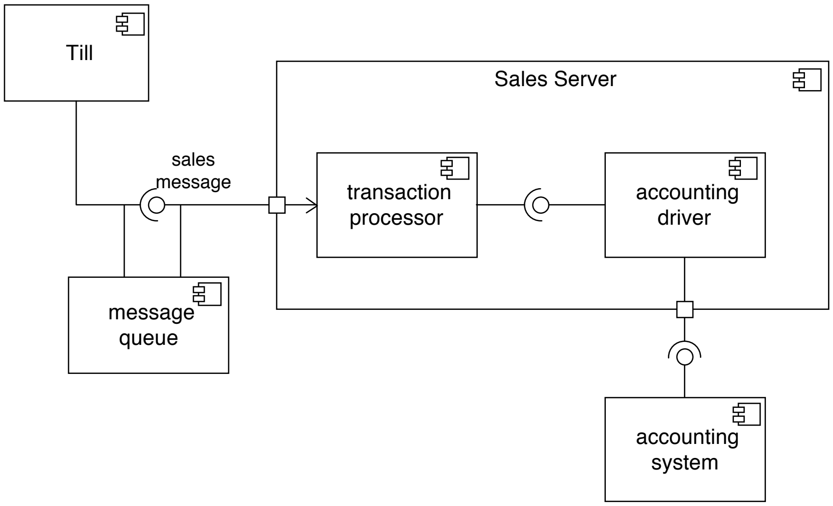
nested packages

package with fully qualified name

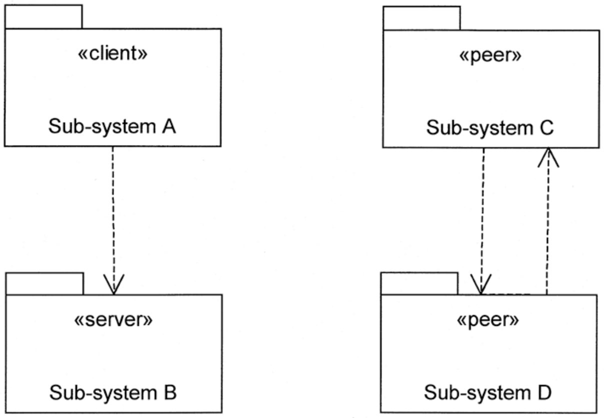
Towards component-based design



Or use Component Diagrams…



Dependency cycles…



The server sub-system does not depend on the client sub-system and is not affected by changes to the client's interface.

Each peer sub-system depends on the other and each is affected by changes in the other's interface.

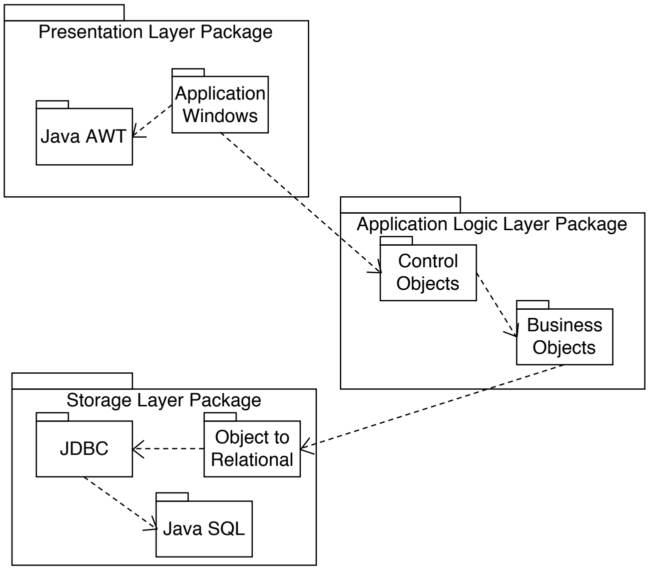
Architectural Patterns

E.g. 3 layer architecture:

Presentation Layer

Application Logic Layer

Storage Layer



Presentation Layer Package

Java AWT

Application Windows

Application Logic Layer Package

Control Objects

Business Objects

Storage Layer Package

JDBC

Object to Relational

Java SQL

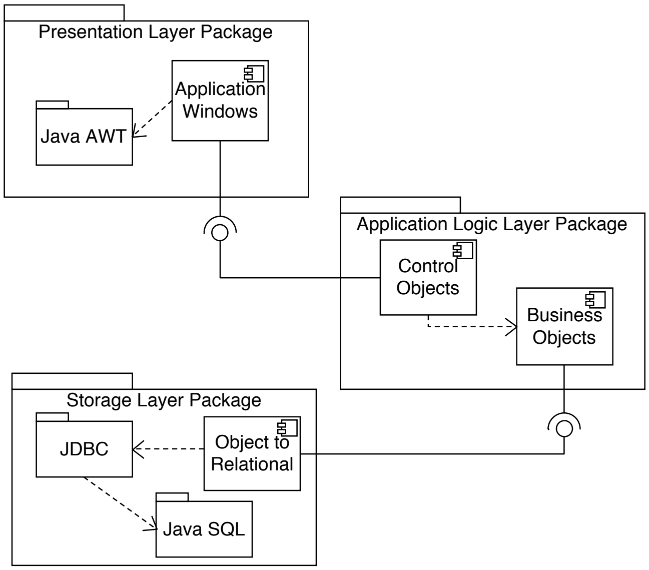
Or to show the interfaces…

E.g. 3 layer architecture:

Presentation Layer

Application Logic Layer

Storage Layer



Presentation Layer Package

Java AWT

Application Windows

Application Logic Layer Package

Control Objects

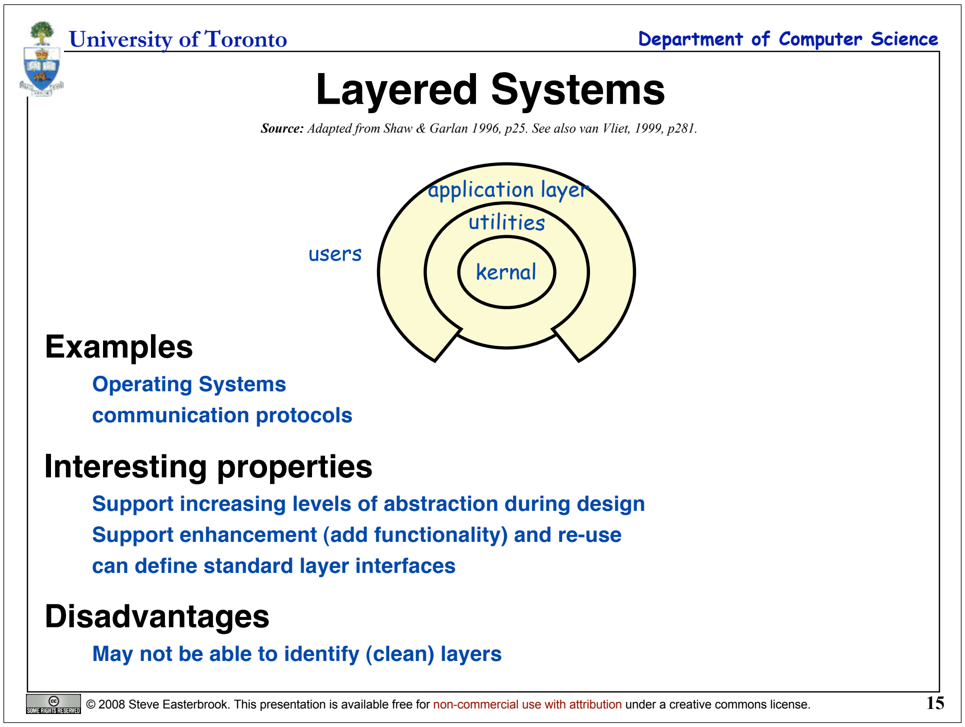
Business Objects

Storage Layer Package

JDBC

Object to Relational

Java SQL



Layered Systems

Source: *Adapted from Shaw & Garlan 1996, p25. See also van Vliet, 1999, p281.*

users
application layer
utilities
kernal

Examples

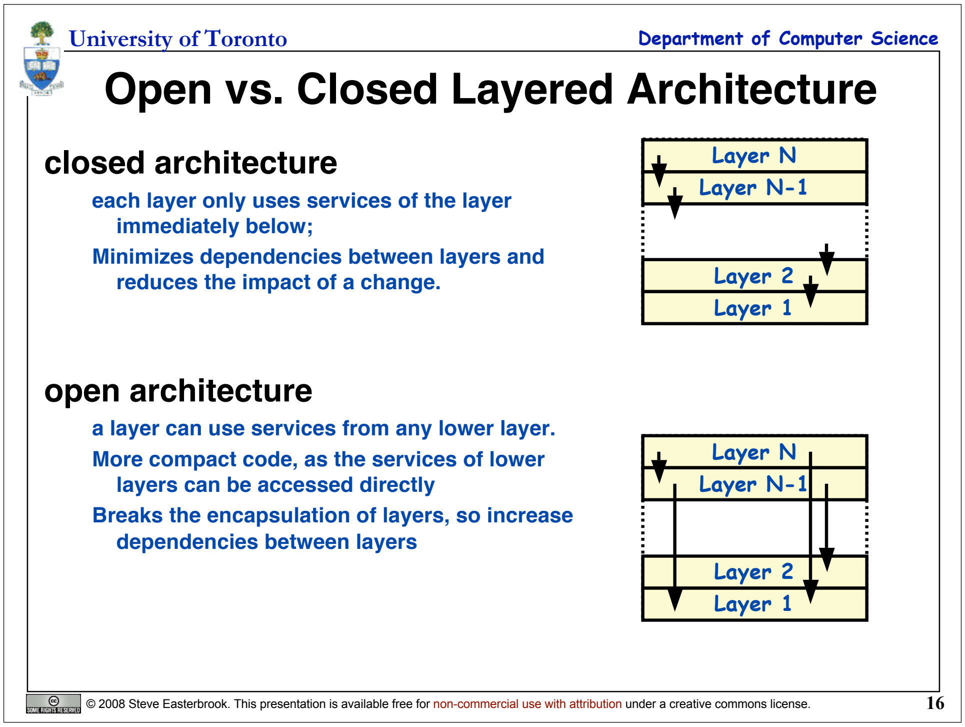
- Operating Systems
- communication protocols

Interesting properties

- Support increasing levels of abstraction during design
- Support enhancement (add functionality) and re-use
- can define standard layer interfaces

Disadvantages

- May not be able to identify (clean) layers



Open vs. Closed Layered Architecture

closed architecture

- each layer only uses services of the layer immediately below;
- Minimizes dependencies between layers and reduces the impact of a change.

Layer N
Layer N-1
Layer 2
Layer 1

open architecture

- a layer can use services from any lower layer.
- More compact code, as the services of lower layers can be accessed directly
- Breaks the encapsulation of layers, so increase dependencies between layers

How many layers?

2-layers:

- application layer
- database layer
- e.g. simple client-server model

3-layers:

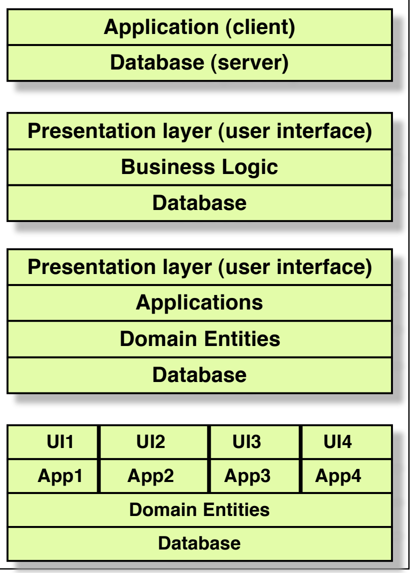
- separate out the business logic
  - helps to make both user interface and database layers modifiable

4-layers:

- Separates applications from the domain entities that they use:
  - boundary classes in presentation layer
  - control classes in application layer
  - entity classes in domain layer

Partitioned 4-layers

- identify separate applications



Application (client)
Database (server)

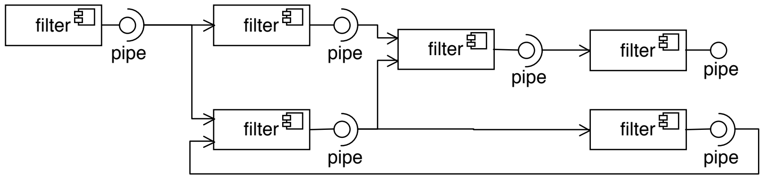
Presentation layer (user interface)
Business Logic
Database

Presentation layer (user interface)
Applications
Domain Entities
Database

UI1	UI2	UI3	UI4
App1	App2	App3	App4
Domain Entities			
Database			

Pipe-and-filter

Source: *Adapted from Shaw & Garlan 1996, p21-2. See also van Vliet, 1999 Pp266-7 and p279*

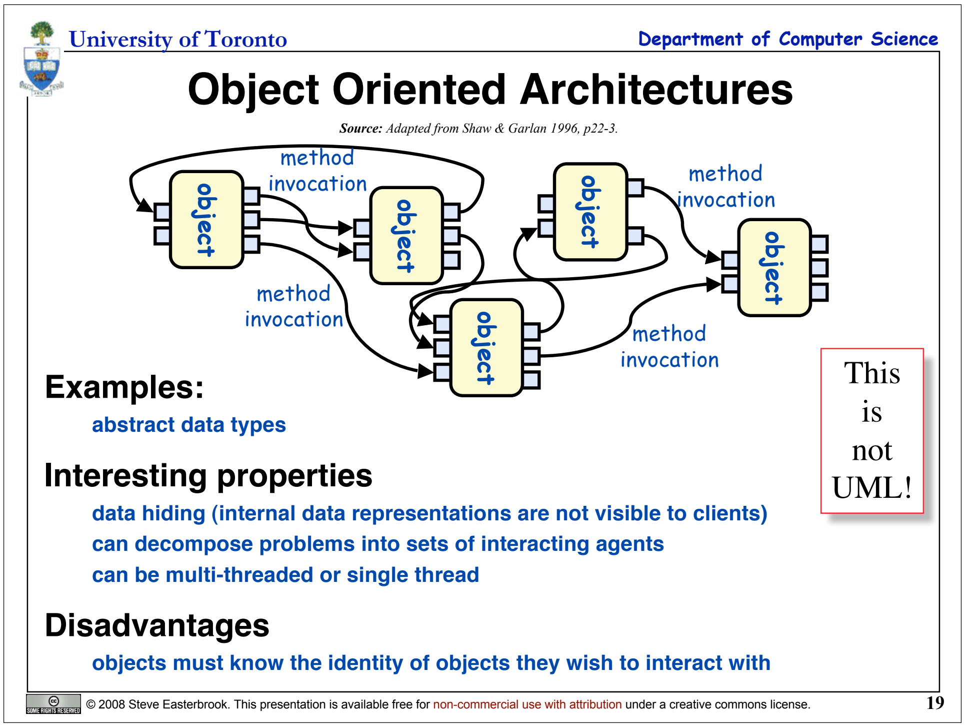


Examples:

- UNIX shell commands
- Compilers:
  - Lexical Analysis -> parsing -> semantic analysis -> code generation
- Signal Processing

Interesting properties:

- filters don’t need to know anything about what they are connected to
- filters can be implemented in parallel
- behaviour of the system is the composition of behaviour of the filters
  - specialized analysis such as throughput and deadlock analysis is possible



Object Oriented Architectures

Source: *Adapted from Shaw & Garlan 1996, p22-3.*

object
object
object
object
object
method invocation
method invocation
method invocation
method invocation

Examples:

abstract data types

Interesting properties

data hiding (internal data representations are not visible to clients)

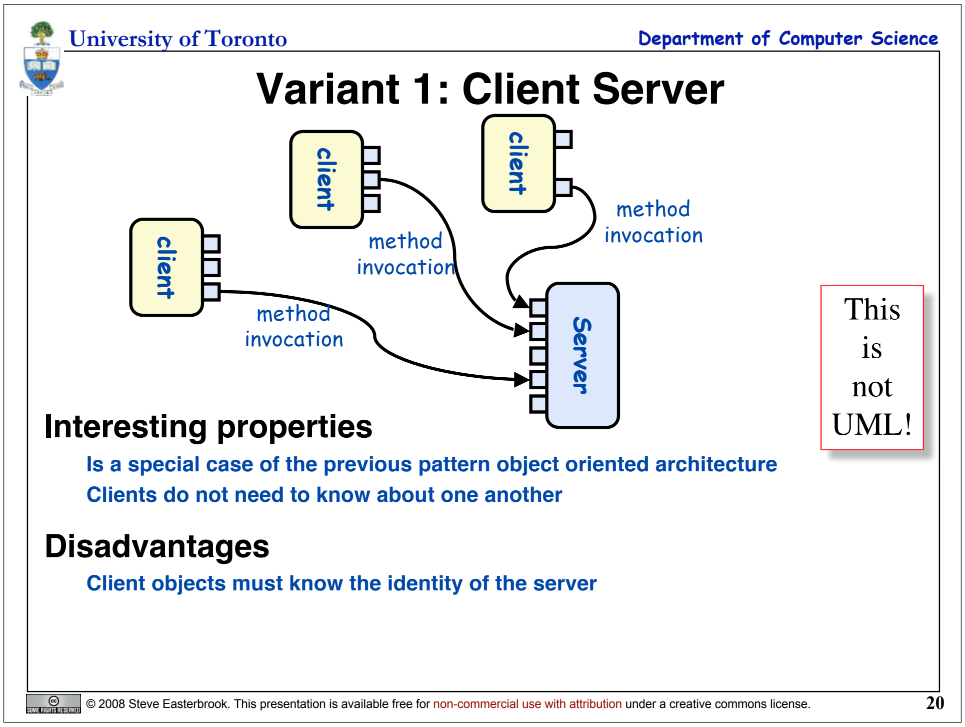
can decompose problems into sets of interacting agents

can be multi-threaded or single thread

Disadvantages

objects must know the identity of objects they wish to interact with

This is not UML!



Variant 1: Client Server

client
client
client
Server
method invocation
method invocation
method invocation

Interesting properties

Is a special case of the previous pattern object oriented architecture

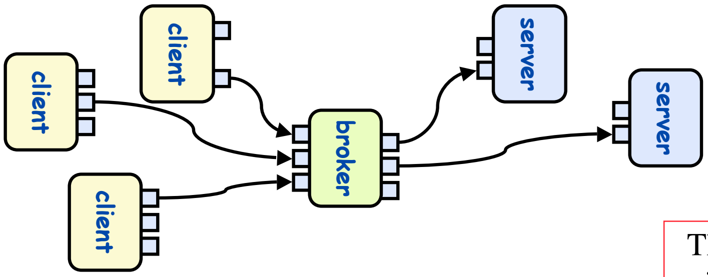
Clients do not need to know about one another

Disadvantages

Client objects must know the identity of the server

This is not UML!

Variant 2: Object Brokers



This is not UML!

Interesting properties

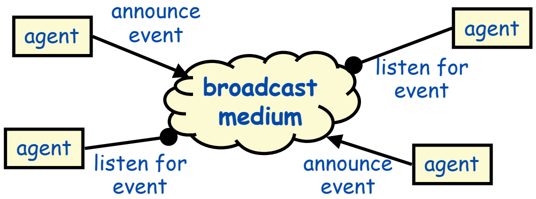
- Adds a broker between the clients and servers
- Clients no longer need to know which server they are using
- Can have many brokers, many servers.

Disadvantages

- Broker can become a bottleneck
- Degraded performance

Event based (implicit invocation)

Source: *Adapted from Shaw & Garlan 1996, p23-4. See also van Vliet, 1999 Pp264-5 and p278*



This is not UML!

Examples

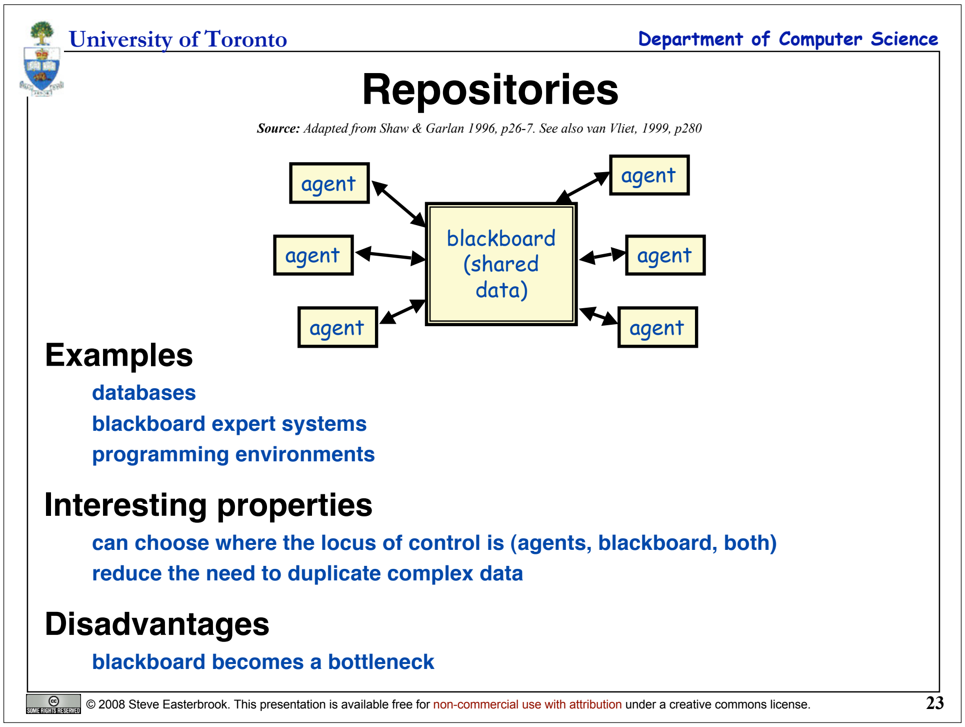
- debugging systems (listen for particular breakpoints)
- database management systems (for data integrity checking)
- graphical user interfaces

Interesting properties

- announcers of events don't need to know who will handle the event
- Supports re-use, and evolution of systems (add new agents easily)

Disadvantages

- Components have no control over ordering of computations



Repositories

Source: *Adapted from Shaw & Garlan 1996, p26-7. See also van Vliet, 1999, p280*

agent
agent
agent
blackboard (shared data)
agent
agent
agent

Examples

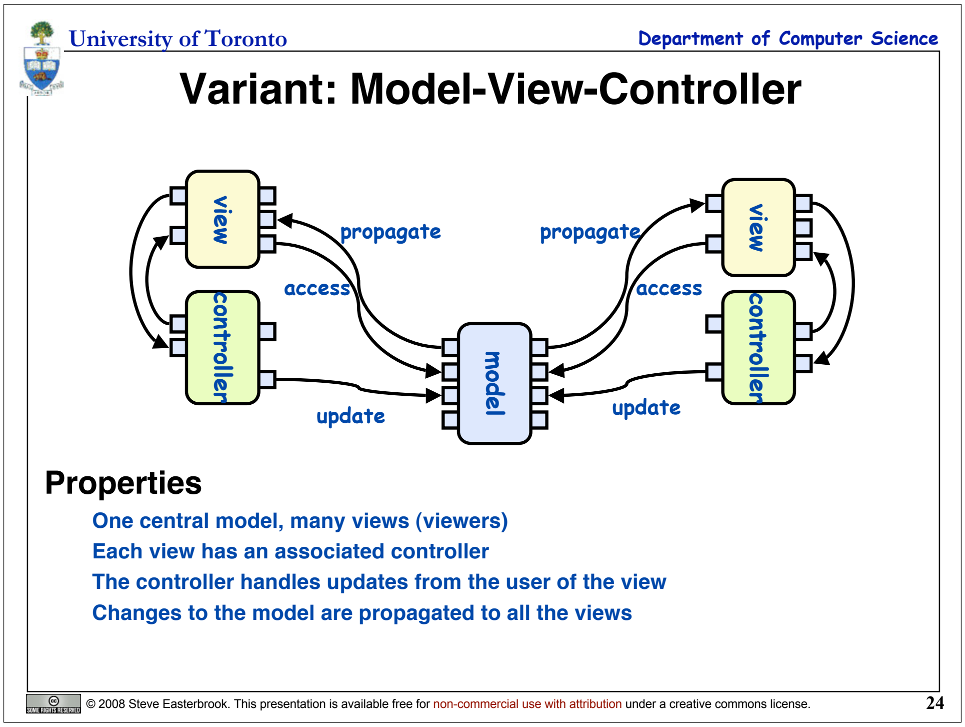
- databases
- blackboard expert systems
- programming environments

Interesting properties

- can choose where the locus of control is (agents, blackboard, both)
- reduce the need to duplicate complex data

Disadvantages

- blackboard becomes a bottleneck



Variant: Model-View-Controller

view
controller
propagate
access
update
model
propagate
access
update
view
controller

Properties

- One central model, many views (viewers)
- Each view has an associated controller
- The controller handles updates from the user of the view
- Changes to the model are propagated to all the views