

University of Toronto Department of Computer Science

## Lecture 13: Software Architectures

⇒ Architectural Styles

- ↳ Pipe and filter
- ↳ Object oriented:
  - > Client-Server; Object Broker
- ↳ Event based
- ↳ Layered:
  - > Designing Layered Architectures
- ↳ Repositories:
  - > Blackboard, MVC
- ↳ Process control

© Easterbrook 2004 1

University of Toronto Department of Computer Science

## 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

© Easterbrook 2004 2

University of Toronto Department of Computer Science

## Object Oriented Architectures

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

⇒ 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

© Easterbrook 2004 3

University of Toronto Department of Computer Science

## Variant 1: Client Server

⇒ 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

© Easterbrook 2004 4

University of Toronto Department of Computer Science

## Variant 2: Object Brokers

```

    graph LR
      C1[client] --> B[broker]
      C2[client] --> B
      C3[client] --> B
      B --> S1[server]
      B --> S2[server]
  
```

- 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

© Easterbrook 2004 5

University of Toronto Department of Computer Science

## Broker Architecture Example

```

    sequenceDiagram
      participant Client as .Client
      participant Proxy as .ClientSide Proxy
      participant Broker as .Broker
      participant ServerProxy as .ServerSide Proxy
      participant Server as .Server

      Client->>Proxy: callServer()
      Proxy->>Broker: sendRequest()
      Broker->>ServerProxy: findServer()
      ServerProxy->>Server: service()
      Server-->>ServerProxy: packData()
      ServerProxy->>Broker: sendResponse()
      Broker->>Proxy: unPackData()
      Proxy-->>Client: sendResponse()
  
```

Possible process boundaries

© Easterbrook 2004 6

University of Toronto Department of Computer Science

## Event based (implicit invocation)

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

```

    graph TD
      A1[agent] -- announce event --> BM((broadcast medium))
      A2[agent] -- listen for event --> BM
      A3[agent] -- announce event --> BM
      A4[agent] -- listen for event --> BM
  
```

- 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

© Easterbrook 2004 7

University of Toronto Department of Computer Science

## Layered Systems

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

```

    graph TD
      Users[users] --- AL[application layer]
      AL --- U[utilities]
      U --- K[kernel]
  
```

- 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

© Easterbrook 2004 8

University of Toronto Department of Computer Science

## Variant: 3-layer data access

© Easterbrook 2004 9

University of Toronto Department of Computer Science

## 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.

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

© Easterbrook 2004 10

University of Toronto Department of Computer Science

## 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

© Easterbrook 2004 11

University of Toronto Department of Computer Science

## Repositories

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

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

© Easterbrook 2004 12

University of Toronto Department of Computer Science

## Variant: Model-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

© Easterbrook 2004 13

University of Toronto Department of Computer Science

## Model View Controller Example

© Easterbrook 2004 14

University of Toronto Department of Computer Science

## MVC Component Interaction

© Easterbrook 2004 15

University of Toronto Department of Computer Science

## Process Control

Source: Adapted from Shaw & Garlan 1996, p27-31.

Examples

- aircraft/spacecraft flight control systems
- controllers for industrial production lines, power stations, etc.
- chemical engineering

Interesting properties

- separates control policy from the controlled process
- handles real-time, reactive computations

Disadvantages

- Difficult to specify the timing characteristics and response to disturbances

© Easterbrook 2004 16