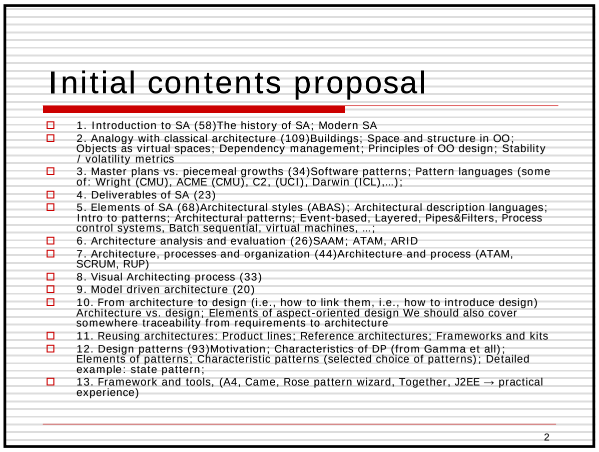
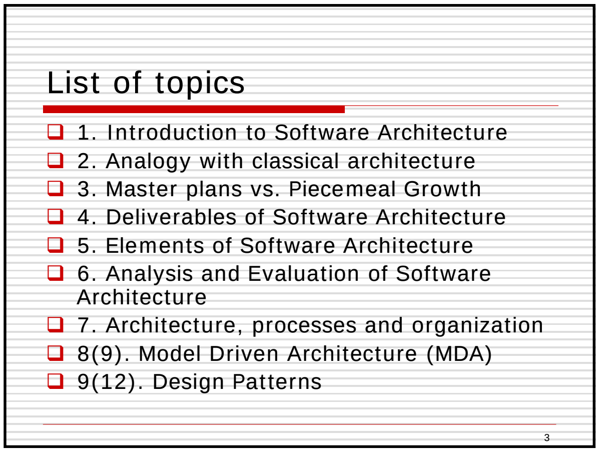
The delivery of the module “Architecture, Design, and Patterns” as part of the Master’s studies in Novi Sad and Skopje

Ioan Jurca (“Politehnica” University of Timisoara - Romania)



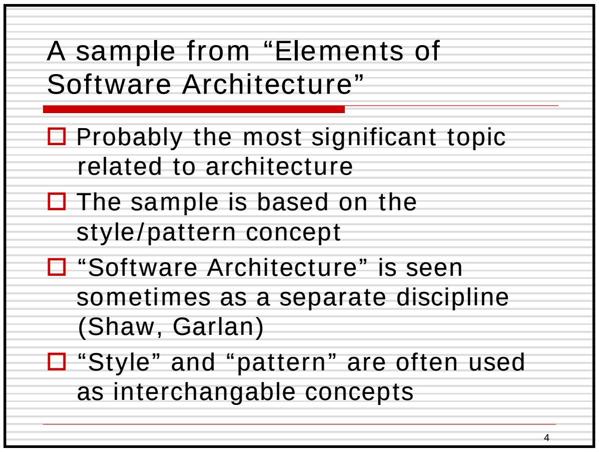
Initial contents proposal

- □ 1. Introduction to SA (58)The history of SA; Modern SA
- □ 2. Analogy with classical architecture (109)Buildings; Space and structure in OO; Objects as virtual spaces; Dependency management; Principles of OO design; Stability / volatility metrics
- □ 3. Master plans vs. piecemeal growths (34)Software patterns; Pattern languages (some of: Wright (CMU), ACME (CMU), C2, (UCI), Darwin (ICL),…);
- □ 4. Deliverables of SA (23)
- □ 5. Elements of SA (68)Architectural styles (ABAS); Architectural description languages; Intro to patterns; Architectural patterns; Event-based, Layered, Pipes&Filters, Process control systems, Batch sequential, virtual machines, …;
- □ 6. Architecture analysis and evaluation (26)SAAM; ATAM, ARID
- □ 7. Architecture, processes and organization (44)Architecture and process (ATAM, SCRUM, RUP)
- □ 8. Visual Architecting process (33)
- □ 9. Model driven architecture (20)
- □ 10. From architecture to design (i.e., how to link them, i.e., how to introduce design) Architecture vs. design; Elements of aspect-oriented design We should also cover somewhere traceability from requirements to architecture
- □ 11. Reusing architectures: Product lines; Reference architectures; Frameworks and kits
- □ 12. Design patterns (93)Motivation; Characteristics of DP (from Gamma et all); Elements of patterns; Characteristic patterns (selected choice of patterns); Detailed example: state pattern;
- □ 13. Framework and tools, (A4, Came, Rose pattern wizard, Together, J2EE → practical experience)



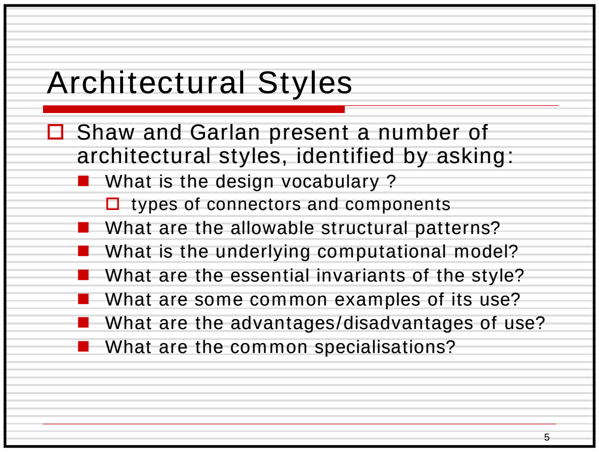
List of topics

- ❑ 1. Introduction to Software Architecture
- ❑ 2. Analogy with classical architecture
- ❑ 3. Master plans vs. Piecemeal Growth
- ❑ 4. Deliverables of Software Architecture
- ❑ 5. Elements of Software Architecture
- ❑ 6. Analysis and Evaluation of Software Architecture
- ❑ 7. Architecture, processes and organization
- ❑ 8(9). Model Driven Architecture (MDA)
- ❑ 9(12). Design Patterns



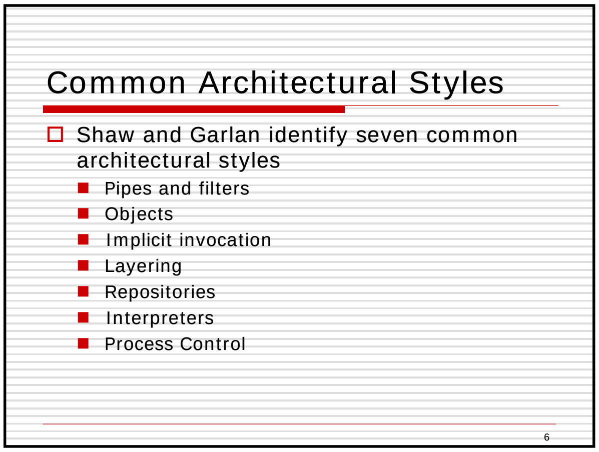
A sample from “Elements of Software Architecture”

- □ Probably the most significant topic related to architecture
- □ The sample is based on the style/pattern concept
- □ “Software Architecture” is seen sometimes as a separate discipline (Shaw, Garlan)
- □ “Style” and “pattern” are often used as interchangable concepts



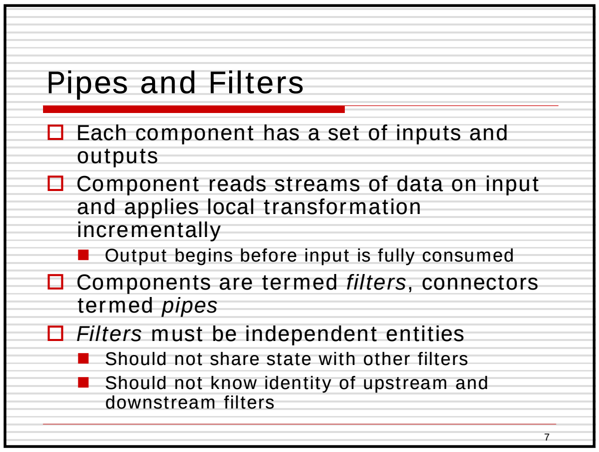
Architectural Styles

- □ Shaw and Garlan present a number of architectural styles, identified by asking:
  - ■ What is the design vocabulary ?
    - □ types of connectors and components
  - ■ What are the allowable structural patterns?
  - ■ What is the underlying computational model?
  - ■ What are the essential invariants of the style?
  - ■ What are some common examples of its use?
  - ■ What are the advantages/disadvantages of use?
  - ■ What are the common specialisations?



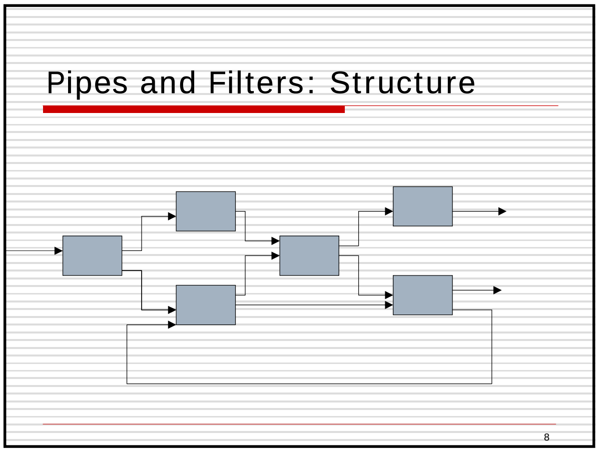
Common Architectural Styles

- □ Shaw and Garlan identify seven common architectural styles
  - ■ Pipes and filters
  - ■ Objects
  - ■ Implicit invocation
  - ■ Layering
  - ■ Repositories
  - ■ Interpreters
  - ■ Process Control

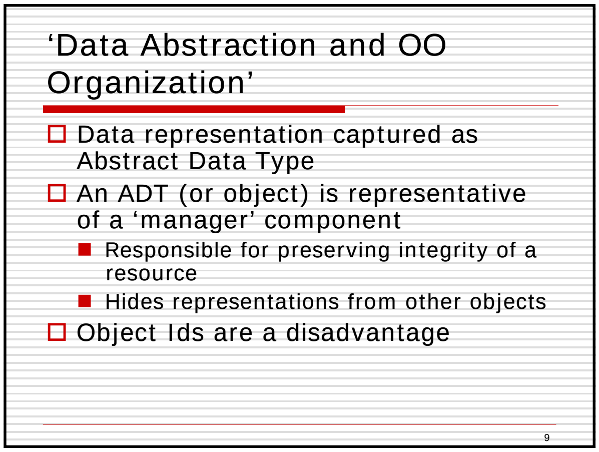


Pipes and Filters

- Each component has a set of inputs and outputs
- Component reads streams of data on input and applies local transformation incrementally
  - Output begins before input is fully consumed
- Components are termed *filters*, connectors termed *pipes*
- *Filters* must be independent entities
  - Should not share state with other filters
  - Should not know identity of upstream and downstream filters

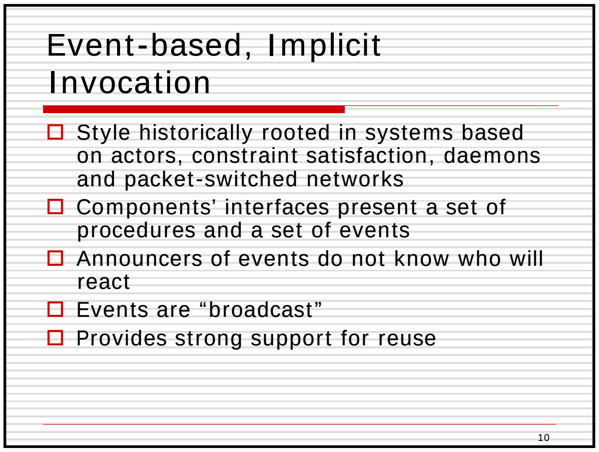


Pipes and Filters: Structure



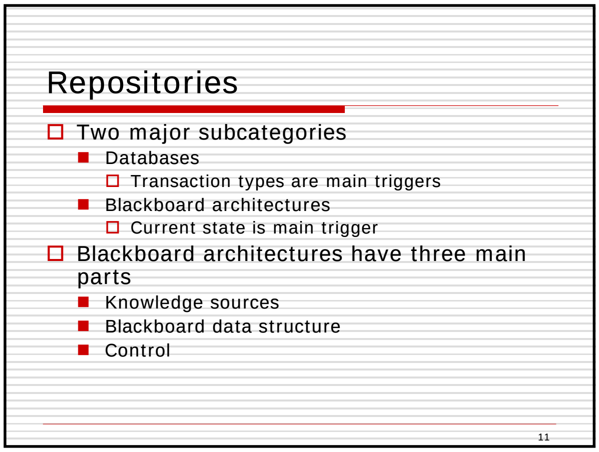
‘Data Abstraction and OO Organization’

- Data representation captured as Abstract Data Type
- An ADT (or object) is representative of a ‘manager’ component
  - Responsible for preserving integrity of a resource
  - Hides representations from other objects
- Object Ids are a disadvantage



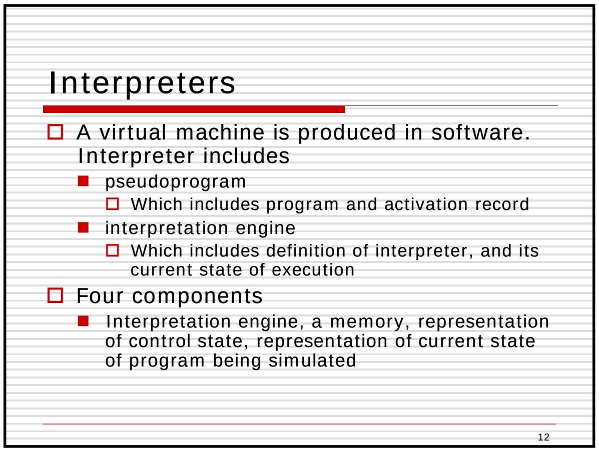
Event-based, Implicit Invocation

- Style historically rooted in systems based on actors, constraint satisfaction, daemons and packet-switched networks
- Components’ interfaces present a set of procedures and a set of events
- Announcers of events do not know who will react
- Events are “broadcast”
- Provides strong support for reuse



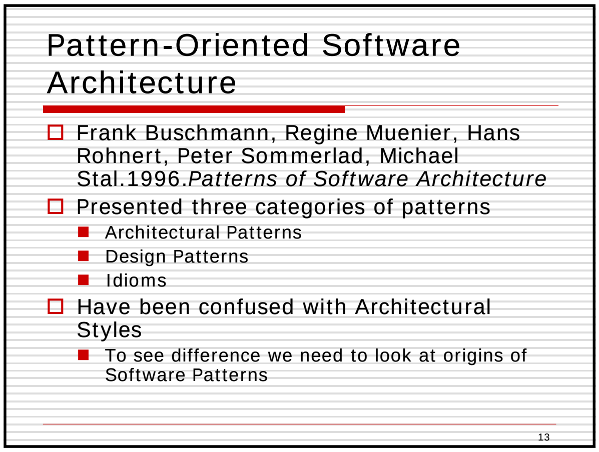
Repositories

- Two major subcategories
  - Databases
    - Transaction types are main triggers
  - Blackboard architectures
    - Current state is main trigger
- Blackboard architectures have three main parts
  - Knowledge sources
  - Blackboard data structure
  - Control



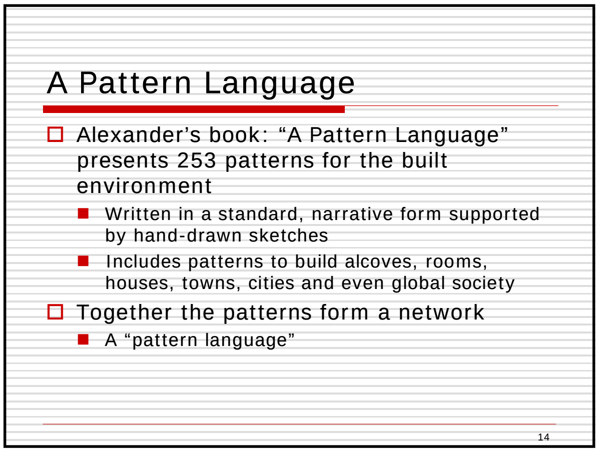
Interpreters

- A virtual machine is produced in software. Interpreter includes
  - pseudoprogram
    - Which includes program and activation record
  - interpretation engine
    - Which includes definition of interpreter, and its current state of execution
- Four components
  - Interpretation engine, a memory, representation of control state, representation of current state of program being simulated



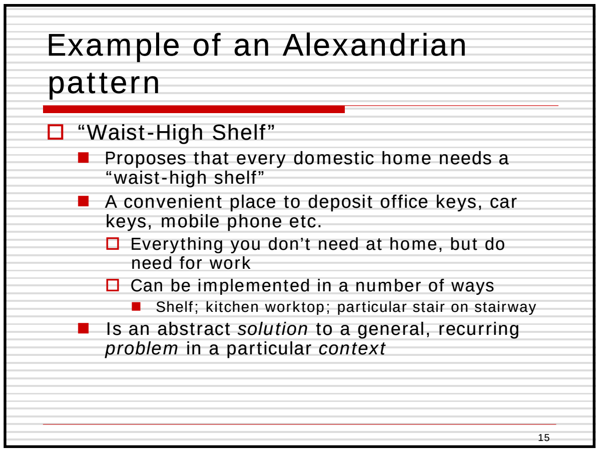
Pattern-Oriented Software Architecture

- □ Frank Buschmann, Regine Muenier, Hans Rohnert, Peter Sommerlad, Michael Stal.1996.*Patterns of Software Architecture*
- □ Presented three categories of patterns
  - ■ Architectural Patterns
  - ■ Design Patterns
  - ■ Idioms
- □ Have been confused with Architectural Styles
  - ■ To see difference we need to look at origins of Software Patterns



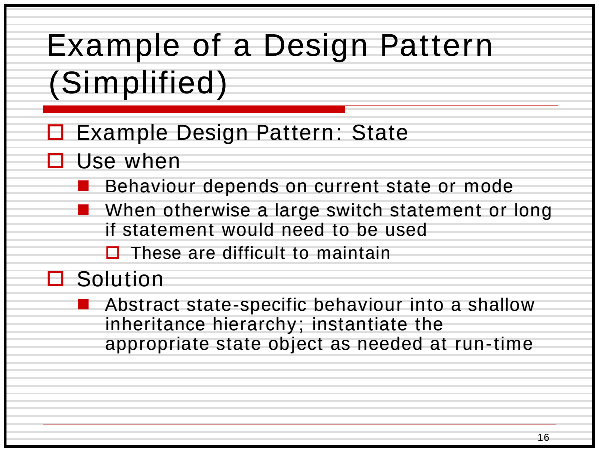
A Pattern Language

- □ Alexander's book: "A Pattern Language" presents 253 patterns for the built environment
  - ■ Written in a standard, narrative form supported by hand-drawn sketches
  - ■ Includes patterns to build alcoves, rooms, houses, towns, cities and even global society
- □ Together the patterns form a network
  - ■ A "pattern language"



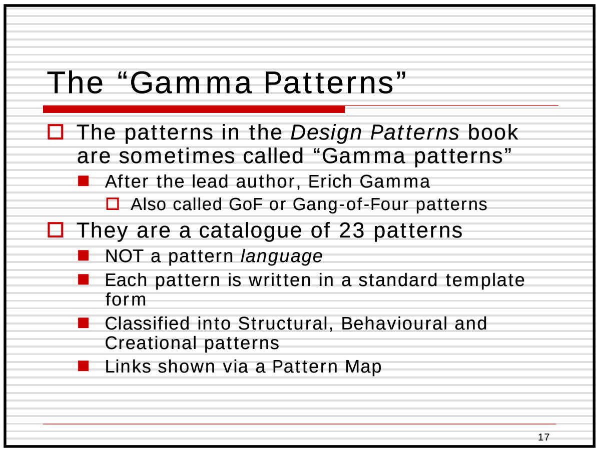
Example of an Alexandrian pattern

- □ "Waist-High Shelf"
  - ■ Proposes that every domestic home needs a "waist-high shelf"
  - ■ A convenient place to deposit office keys, car keys, mobile phone etc.
    - □ Everything you don't need at home, but do need for work
    - □ Can be implemented in a number of ways
      - ■ Shelf; kitchen worktop; particular stair on stairway
  - ■ Is an abstract *solution* to a general, recurring *problem* in a particular *context*



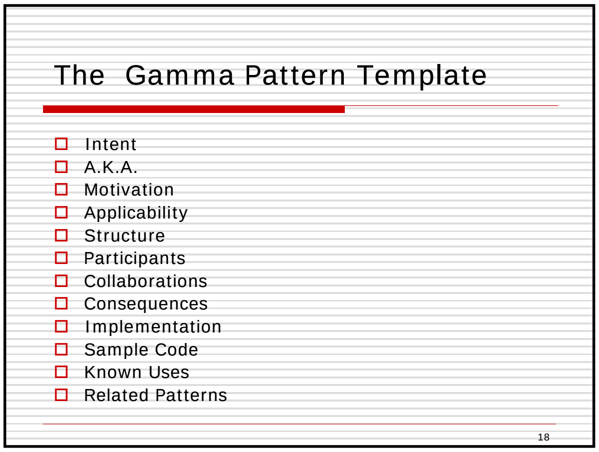
Example of a Design Pattern (Simplified)

- □ Example Design Pattern: State
- □ Use when
  - ■ Behaviour depends on current state or mode
  - ■ When otherwise a large switch statement or long if statement would need to be used
    - □ These are difficult to maintain
- □ Solution
  - ■ Abstract state-specific behaviour into a shallow inheritance hierarchy; instantiate the appropriate state object as needed at run-time



The "Gamma Patterns"

- □ The patterns in the *Design Patterns* book are sometimes called "Gamma patterns"
  - ■ After the lead author, Erich Gamma
    - □ Also called GoF or Gang-of-Four patterns
- □ They are a catalogue of 23 patterns
  - ■ NOT a pattern *language*
  - ■ Each pattern is written in a standard template form
  - ■ Classified into Structural, Behavioural and Creational patterns
  - ■ Links shown via a Pattern Map



The Gamma Pattern Template

- □ Intent
- □ A.K.A.
- □ Motivation
- □ Applicability
- □ Structure
- □ Participants
- □ Collaborations
- □ Consequences
- □ Implementation
- □ Sample Code
- □ Known Uses
- □ Related Patterns

Characteristics of Software Design Patterns (e.g. Gamma et al)*

- □ Problem, not solution-centred
- □ Focus on "non-functional" aspects
- □ Discovered, not invented
- □ Complement, do not replace existing techniques
- □ Proven record in capturing, communicating "best practice" design expertise

*Gamma E., Helm R., Johnson R., Vlissides J. 1994.***Design Patterns- Elements of Reusable Object-Oriented Software.*** Addison-Wesley

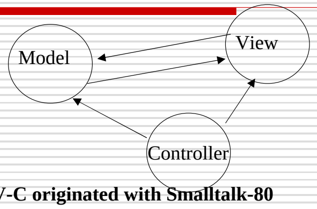
Architectural Patterns

- □ "An architectural pattern expresses a fundamental organising structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them" (p.12)

Architectural Patterns

- □ Buschmann et al., present a catalogue that includes 8 architectural patterns in 4 categories
  - ■ "From Mud to Structure"
    - □ Layers, Pipes and Filters, Blackboard
  - ■ Distributed Systems
    - □ Broker
  - ■ Interactive Systems
    - □ Model View Controller, Presentation-Abstraction-Controller
  - ■ Adaptable Systems
    - □ Microkernel, Reflection

The Model-View-Controller Pattern



•**M-V-C originated with Smalltalk-80**

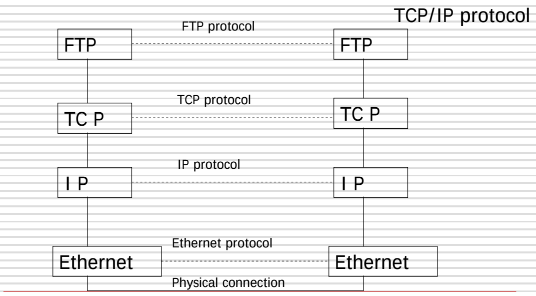
- Informs the entire architecture of modern Smalltalk environments

•**Microsoft's Document-View architecture is an instance of M-V-C**

•Model = Document, View = View

-So where is the Controller? (answer: it is MS Windows!)

Layers Pattern:Example



Layers Pattern

- □ Context
  - ■ large system needing decomposition
- □ Problem
  - ■ How to structure systems that contain a mix of high and low-level functionality
- □ Solution
  - ■ Conceptually layer the system, from level 0 upwards

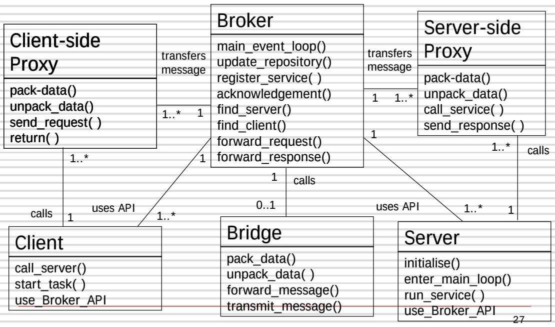
Layers Pattern: Consequences

- Benefits
  - Reuse of Layers
  - Support for standardisation
  - Localisation of dependencies
  - Exchangeability
- Liabilities
  - Cascades of Changing Behaviour
  - Lower Efficiency
  - Unnecessary work
  - Difficulty of getting ‘granularity’ right

Broker Pattern

- Context
  - Distributed, possibly heterogeneous system of independent co-operating “components”
- Problem
  - How to partition functionality to deliver a set of decoupled, interoperating components
- Solution
  - Introduce a Broker component to decouple clients and servers

Broker Pattern:Structure



Client-side Proxy
pack-data()
unpack_data()
send_request( )
return( )

Broker
main_event_loop()
update_repository()
register_service( )
acknowledgement()
find_server()
find_client()
forward_request()
forward_response()

Server-side Proxy
pack-data()
unpack_data()
call_service( )
send_response( )

Client
call_server()
start_task( )
use_Broker_API

Bridge
pack_data()
unpack_data( )
forward_message()
transmit_message()

Server
initialise()
enter_main_loop()
run_service( )
use_Broker_API

Broker Pattern: Variants

- Direct Communication Broker System
  - Clients communicate directly with servers, broker identifies the communication channel
- Message Passing Broker System
  - Servers use type of message to determine action
- Trader System
  - Client-side servers provide *service* ids rather than *server* ids
- Adapter Broker System
- Callback Broker System
  - Reactive, event-driven model; makes no distinction between clients and servers

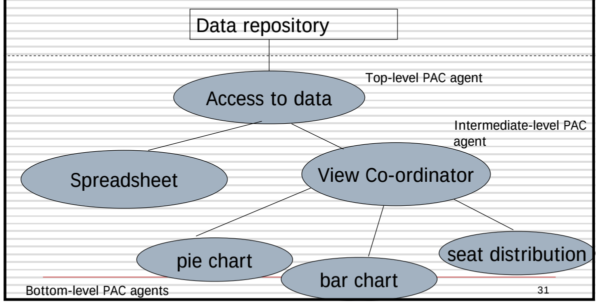
Broker Pattern: Consequences

- Benefits
  - Location transparency
  - Changeability/Extensibility of components
  - Portability
  - Interoperability between Broker Systems
  - Reusability
  - Testing and Debugging
- Liabilities
  - Restricted efficiency
  - Lower fault tolerance
  - Testing and Debugging

Presentation-Abstraction-Control

- Context
  - Interactive systems with the help of agents
- Problem
  - Partitioning of interactive systems horizontally and vertically
- Solution
  - Structure the solution as a tree-like hierarchy of PAC agents

Presentation-Abstraction-Control: Structure



Presentation-Abstraction-Control: Consequences

- Benefits
  - Separation of Concerns
  - Support for Change/Extension
  - Support for multi-tasking
- Liabilities
  - Increased system complexity
  - Complex control components
  - Efficiency
  - Restricted applicability

Delivery in Novi Sad

- Two weekends (in March and April)
- Total delivery hours:20
- Attendance: 12-15 students from Novi Sad and Nis
- Not accompanied by exercises
- Lectures recorded
- Small number of questions from the students

Delivery in Skopje

- One weekend (in April)
- Total delivery hours:16
- Attendance: 12-15 students from Skopje
- Not accompanied by exercises
- Some topics covered only summarily
- Reasonable number of questions from the students

A few conclusions (1)

- New topics have to be developed over the summer/atumn
- 20 hours for lectures is not enough to cover in-depth all topics
- Students involvement during lectures must be increased
- Development of assignments: first attempt can be study and reporting of ‘classical papers’
- Desirable assignment: analysis and critics of the architecture of an open-source application of medium size

A few conclusions (2)

- I would like to continue involvement in developing the module
- The relation between requirements and architecture is an important topic
- There is considerable research interest in this topic
- Patterns can be separated into a ‘stand-alone’ module, possibly covering all types of *software patterns*