System Design and Architecture

Based on Chapter 13
Bennett, McRobb and Farmer

Object Oriented Systems Analysis and Design Using UML
In This Lecture You Will Learn:

- the major concerns of system design
- what is meant by architecture in information systems development
- the factors that influence the architecture of a system
In This Lecture You Will Learn:

• the range of architectural styles that can be used, including layers and partitions
• how to apply the Model–View–Controller architecture
• which architectures are suitable for distributed systems.
Architecture

Architects are trained to take your brief and can see the big picture – they look beyond your immediate requirements to design flexible buildings that will adapt with the changing needs of your business.

Architects solve problems creatively – when they are involved at the earliest planning stage, they gain more opportunities to understand your business, develop creative solutions, and propose ways to reduce costs.

RIBA (2005)
System Architecture

• System architects:
  – act on behalf of the client;
  – address the big picture;
  – ensure that the required qualities of the system are accounted for in the design;
  – solve problems;
  – ensure the required features are provided at the right cost.
System Architecture

Key Definitions

• **System** is a set of components that accomplishes a specific function or set of functions.

• **Architecture** is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.

• **Architectural Description** is a set of products that document the architecture.
System Architecture

Key Definitions

• *Architectural View* is a representation of a particular system or part of a system from a particular perspective.

• *Architectural Viewpoint* is a template that describes how to create and use an architectural view. A viewpoint includes a name, stakeholders, concerns addressed by the viewpoint, and the modelling and analytic conventions.

(Garland & Anthony, 2003 & IEEE, 2000)
System Architecture

Architecture Types

- Soni et al.

<table>
<thead>
<tr>
<th>Type of architecture</th>
<th>Examples of elements</th>
<th>Examples of relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual</td>
<td>Components</td>
<td>Connectors</td>
</tr>
<tr>
<td>Module</td>
<td>Subsystems, modules</td>
<td>Exports, imports</td>
</tr>
<tr>
<td>Code</td>
<td>Files, directories, libraries</td>
<td>Includes, contains</td>
</tr>
<tr>
<td>Execution</td>
<td>Tasks, threads, object interactions</td>
<td>Uses, calls</td>
</tr>
</tbody>
</table>

Adapted from Weir and Daniels (1998)
## System Architecture

### Architecture Views

- **Kruchten - RUP’s 4 = 1 views**

<table>
<thead>
<tr>
<th>View</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use case view</td>
<td>The important use cases in the system and scenarios that describe architecturally significant behaviour.</td>
</tr>
<tr>
<td>Logical view</td>
<td>Important design classes and interfaces in a package structure, with composite structure diagrams.</td>
</tr>
<tr>
<td>Implementation view</td>
<td>Architectural decisions made for the implementation in terms of subsystems and components and relationships among them.</td>
</tr>
<tr>
<td>Process view</td>
<td>A description of the processes (operating system processes and threads) and inter-process communications using stereotyped classes.</td>
</tr>
<tr>
<td>Deployment view</td>
<td>Physical nodes for the likely deployment platform, components deployed on the nodes and the communication channels between them, using deployment diagrams.</td>
</tr>
</tbody>
</table>
System Architecture

ADL

• Architecture Description Language
  – UML 2.0 adds or changes features to support modelling architecture
  – Package diagrams
  – Component diagrams
  – Composite structure diagrams
  – Deployment diagrams
System Architecture

Why model it?

• Architects use models to reason about the system and its ability to deliver the required quality attributes (reliability, performance, security etc.).
• Particular views help to reason about particular quality attributes.
## System Architecture: Why model it?

<table>
<thead>
<tr>
<th>View</th>
<th>Contribution to assessing performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use case view</td>
<td>The use cases that require high performance can be identified and the scenarios used to walkthrough how the other views will affect the performance requirement.</td>
</tr>
<tr>
<td>Logical view</td>
<td>The logical view of classes will show whether techniques such as creating lightweight objects or value objects have been used to reduce the overheads associated with passing values around.</td>
</tr>
<tr>
<td>Implementation view</td>
<td>The more components or subsystems are involved, the more likely there are to be communication overheads, so the implementation view should show a small number of components used in the process.</td>
</tr>
<tr>
<td>Process view</td>
<td>The process view can be used to assess how many running processes will exist, and whether there will be multiple instances of the same process so that the work can be shared out by a special process that handles load-balancing. The kind of inter-process communication that is used will affect how efficiently data can be passed between processes.</td>
</tr>
<tr>
<td>Deployment view</td>
<td>The deployment view will show where different components are deployed, and whether data has to travel from machine to machine, or whether all the processes needed to deliver a high-performance use case are located on the same machine.</td>
</tr>
</tbody>
</table>
System Architecture
Influences

• Existing systems
• Enterprise architectures
• Technical reference architectures
System Architecture Influences

- **Existing Systems**
  - Standard architectures

- **Heritage systems**
  - May be wrapped using adapters

- **Services**
  - Wrapping systems with adapters

- **Reverse-engineering and MDA**
  - Generate platform-specific models (PSMs) from platform-independent models (PIMs)
  - Reverse-engineer existing system logic to PIMs
System Architecture Influences

• Enterprise Architectures
  – Pressure in the United States
    • Clinger-Cohen Act 1996
    • Sarbanes-Oxley Act 2002
  – Frameworks
    • Federal Enterprise Architecture Framework
    • Standards and Architectures for eGovernment Applications
    • Zachman Framework
System Architecture Influences

• Technical Reference Architectures
  – Standards for technologies to apply
  – Guidance on how to apply them

• The Open Group Architecture Framework (TOGAF)
  – Architecture Development Method
  – Enterprise Continuum
  – Resources
System Architecture

Architectural Styles

- Architectural styles apply to physical architecture and to software architecture
- Bass et al identify five main types:
  - Independent components
  - Data flow
  - Data centered
  - Virtual machine
  - Call and return
Subsystems

• A subsystem typically groups together elements of the system that share some common properties

• An object-oriented subsystem encapsulates a coherent set of responsibilities in order to ensure that it has integrity and can be maintained
Subsystems

• The subdivision of an information system into subsystems has the following advantages
  – It produces smaller units of development
  – It helps to maximize reuse at the component level
  – It helps the developers to cope with complexity
  – It improves maintainability
  – It aids portability
Subsystems

• Each subsystem provides services for other subsystems, and there are two different styles of communication that make this possible

• These are known as *client–server* and *peer-to-peer* communication
Styles of communication between subsystems

The server subsystem does not depend on the client subsystem and is not affected by changes to the client’s interface.

Each peer subsystem depends on the other and each is affected by changes in the other’s interface.
Client–server communication

- Client–server communication requires the client to know the interface of the server subsystem, but the communication is only in one direction.
- The client subsystem requests services from the server subsystem and not vice versa.
Peer-to-peer communication

• Peer-to-peer communication requires each subsystem to know the interface of the other, thus coupling them more tightly

• The communication is two way since either peer subsystem may request services from the other
Layering and Partitioning

• Two general approaches to the division of a software system into subsystems
  – *Layering*—so called because the different subsystems usually represent different levels of abstraction
  – *Partitioning*, which usually means that each subsystem focuses on a different aspect of the functionality of the system as a whole

• Both approaches are often used together on one system
Schematic of a Layered Architecture

Closed architecture—messages may only be sent to the adjacent lower layer.

Open architecture—messages can be sent to any lower layer.
Layered Architecture

• A closed architecture minimizes dependencies between the layers and reduces the impact of a change to the interface of any one layer.

• An open layered architecture produces more compact code, the services of all lower level layers can be accessed directly by any layer above them without the need for extra program code to pass messages through each intervening layer, but this breaks the encapsulation of the layers.
<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7: Application</td>
<td>Provides miscellaneous protocols for common activities.</td>
</tr>
<tr>
<td>6: Presentation</td>
<td>Structures information and attaches semantics.</td>
</tr>
<tr>
<td>5: Session</td>
<td>Provides dialogue control and synchronization facilities.</td>
</tr>
<tr>
<td>4: Transport</td>
<td>Breaks messages into packets and ensures delivery.</td>
</tr>
<tr>
<td>3: Network</td>
<td>Selects a route from sender to receiver.</td>
</tr>
<tr>
<td>2: Data Link</td>
<td>Detects and corrects errors in bit sequences.</td>
</tr>
<tr>
<td>1: Physical</td>
<td>Transmits bits: sets transmission rate (baud), bit-code, connection, etc.</td>
</tr>
</tbody>
</table>
Applying a Layered Architecture

• Issues that need to be addressed include:
  – maintaining the stability of the interfaces of each layer
  – the construction of other systems using some of the lower layers
  – variations in the appropriate level of granularity for subsystems
  – the further sub-division of complex layers
  – performance reductions due to a closed layered architecture

(Buschmann et al., 1996)
Simple Layered Architecture.

Application

↓

Data formatting

↓

Data management
Developing a Layered Architecture

1. Define the criteria by which the application will be grouped into layers. A commonly used criterion is level of abstraction from the hardware.
2. Determine the number of layers.
3. Name the layers and assign functionality to them.
4. Specify the services for each layer.
5. Refine the layering by iterating through steps 1 to 4.
Developing a Layered Architecture

6. Specify interfaces for each layer.
7. Specify the structure of each layer. This may involve partitioning within the layer.
8. Specify the communication between adjacent layers (this assumes that a closed layer architecture is intended).
9. Reduce the coupling between adjacent layers. This effectively means that each layer should be strongly encapsulated.

(Adapted from Buschmann et al., 1996)
Three & Four Layer Architectures.

- Presentation
- Business logic
- Database

- Presentation
- Application logic
- Domain
- Database

Business logic layer can be split into two layers.
Partitioned Subsystems

Loosely coupled subsystems, each delivering a single service or coherent group of services.

Four layer architecture applied to part of the Agate campaign management system.

A single domain layer supports two application subsystems.

Presentation layer

- Advert HCI Subsystem
- Advert Subsystem

Application layer

- Campaign Costs HCI Subsystem
- Campaign Costs Subsystem

- Campaign Domain
- Campaign Database
Problems with some Architectures

Each subsystem contains some core functionality

Changes to data in one subsystem need to be propagated to the others

Campaign Forecasting

Advert Development

Campaign Management

Campaign and Advert Database Access

Multiple interfaces for the same core functionality.
Difficulties

• We repeat below some of the difficulties that need to be resolved for this type of application
  – The same information should be capable of presentation in different formats in different windows
  – Changes made within one view should be reflected immediately in the other views
  – Changes in the user interface should be easy to make
  – Core functionality should be independent of the interface to enable multiple interface styles to co-exist
Model-View-Controller

(Adapted from Hopkins and Horan, 1995)

© 2010 Bennett, McRobb and Farmer
Model-View-Controller

- **Model**—provides the central functionality of the application and is aware of each of its dependent view and controller components.
- **View**—corresponds to a particular style and format of presentation of information to the user. The view retrieves data from the model and updates its presentations when data has been changed in one of the other views. The view creates its associated controller.
Model-View-Controller

- **Controller**—accepts user input in the form of events that trigger the execution of operations within the model. These may cause changes to the information and in turn trigger updates in all the views ensuring that they are all up to date.

- **Propagation Mechanism**—enables the model to inform each view that the model data has changed and as a result the view must update itself. It is also often called the dependency mechanism.
MVC applied to Agate

Navigability arrows show the directions in which messages will be sent.

**CampaignModel**
- CoreData
- setOfObservers [0..*]
- attach(Observer)
- detach(Observer)
- notify()
- getAdvertData()
- modifyAdvert()

**AdvertView**
- initialize()
- displayAdvert()
- update()

**AdvertController**
- initialize()
- changeAdvert()
- update()
MVC Component Interaction

sd Change advert

:AdvertController

:CampaignModel

:AdvertView

changeAdvert

modifyAdvert

notify

update

displayAdvert

getAdvertData

update

good
Schematic of Simplified Broker Architecture

- «component» Client A
- «component» Client B
- «component» Broker
- «component» Server 1
- «component» Server 2
- «component» Server 3

© 2010 Bennett, McRobb and Farmer
Broker-based client–server communication

(Client) callServer

(ClientSide Proxy) sendRequest → unpackData

(Broker) findServer

(ServerSide Proxy) requestService → unpackData

(Server) sendResponse
Process Allocation

- **ClientProcess**
  - components:
    - Client
    - ClientSideProxy

- **BrokerProcess**
  - components:
    - Broker

- **ServerProcess**
  - components:
    - Server
    - ServerSideProxy
Schematic of broker architecture using bridge components
Concurrent activity in an interaction diagram

- **sd** Concurrent execution
- **par** Asynchronous messages
- **msg a**
- **msg b**
- **msg c**
- **msg d**

- Simultaneous execution
- Do not execute simultaneously
Scheduler Handling
Concurrency

This thread of execution generates a system output.

Thread of control invoked by scheduler and produces no output.

Interrupts generated in scheduler.
Processor Allocation

• Allocation of a system to multiple processors
  – Application should be divided into subsystems
  – Estimate processing requirements for subsystems
  – Determine access criteria and location requirements
  – Identify concurrency requirements
  – Each subsystem should be allocated to an operating platform
  – Communication requirements between subsystems should be determined
  – The communications infrastructure should be specified
Planning for Design

- Planning for when platform is known
- Setting standards
- Allowing time for training
- Agreeing objectives and planning tests
- Agree procedures to decide on trade-offs that significantly affect the system
- Planning time for different aspects of design
Development Standards

• HCI guidelines
• Input/output device guidelines
• Construction guidelines
I/O Device Hierarchy

I/O Hierarchy providing consistency for device handling

- IODevice
  - handleEvent()
  - Sensor
    - handleEvent()
  - Actuator
    - handleEvent()
Prioritizing Design Trade-offs

- Designer is often faced with design objectives that are mutually incompatible.
- It is helpful if guidelines are prepared for prioritizing design objectives.
- If design choice is unclear users should be consulted.
Trade-offs in Design

• Design to meet all these qualities may produce conflicts
• Trade-offs have to be applied to resolve these
• Functionality, reliability and security are likely to conflict with economy
• Level of reliability, for example, is constrained by the budget available for the development of the system
Possible package architecture for Agate, showing how it will be implemented as Java packages or C# namespaces.
### Agate Case Study

<table>
<thead>
<tr>
<th>«process» AgateClientProcess</th>
<th>«process» AgateServerProcess</th>
</tr>
</thead>
<tbody>
<tr>
<td>components</td>
<td>components</td>
</tr>
<tr>
<td>com.agate.boundary</td>
<td>com.agate.control.server</td>
</tr>
<tr>
<td>com.agate.control.client</td>
<td>com.agate.domain.bo</td>
</tr>
<tr>
<td>com.agate.domain.vo</td>
<td>com.agate.domain.vo</td>
</tr>
<tr>
<td></td>
<td>com.agate.database</td>
</tr>
</tbody>
</table>
Summary

In this lecture you have learned about:

• The main aspects of system architecture, in particular what is meant by subdividing a system into layers and partitions
• How to apply the MVC architecture
• Which architectures are most suitable for distributed systems
References

• Buschmann et al. (1996)
• Bass et al. (2003)
• Garland and Anthony (2003)
  (For full bibliographic details, see Bennett, McRobb and Farmer)
Summary

• In this lecture you have learned about:
  – the major concerns of system design
  – what is meant by architecture in information systems development
  – the factors that influence the architecture of a system
Summary

In this lecture you have learned about:

- the range of architectural styles that can be used, including layers and partitions
- how to apply the Model–View–Controller architecture
- which architectures are suitable for distributed systems.
References

• Bass et al. (2003)
• Garland and Anthony (2003)
• Buschmann et al. (1996).
• Kruchten (2004)
  (For full bibliographic details, see Bennett, McRobb and Farmer)