

Software Architecture and Architectural Styles

Lecture 5

U08182 © Peter Lo 2010

1

In this lecture you will learn:

- Software architecture
 - Introduction to the notion
 - Prescriptive view
 - Descriptive view
 - Role in software design
- Description of software architecture
 - A diagrammatic representation notation
- Software architectural styles
 - Introduction to the notion
 - Some typical software architectural styles
 - Catalogue of software architectural style
- Use of software architectural styles in design
- Analysis of software architectural designs

Prescriptive Point of View

- **Software Architecture** is regarded as a design document that prescribes how the system is to be built and its rationale.

“Architecture is concerned with the selection of architectural elements, their interactions, and the constraints on those elements and their interactions necessary to provide a framework in which to satisfy the requirements and serve as a basis for the (detail) design.”

Perry and Wolf, 1992

Software Architecture = {Elements, Form, Rationale}

U08182 © Peter Lo 2010

2

Architectural Elements

- **Data Elements** contain the information that is used and transformed
- **Processing Elements** supply the transformation on the data elements
- **Connecting Elements** are the 'glue' that holds the different pieces of the architecture together, for example, procedure calls, accesses to shared data, and messages

Architectural Form

- Consists of weighted properties and relationships.
 - **Relationships** constrain how the different elements may interact and how they are organised with respect to each other in the architecture.
 - **Properties** are used to constrain the choice of architectural elements.
 - **Weight** can be associate to a property or relationship to indicate their importance, or to express the preference among a number of choices among alternatives.

The Rationale

- It captures the motivation for the choice of architectural style, the choice of elements, and the form.
- It explicates the satisfaction of system constraints determined by users' requirements both functional and non-functional.

Descriptive Point of View

- **Software Architecture** is considered as a description of the high level structure of a software system in terms of architectural elements and the interactions between them.

“Abstractly, software architecture involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on these patterns.”

Shaw and Garlan, 1996

Software Architecture = {Components, Connectors}

U08182 © Peter Lo 2010

3

Components

- A component is a unit of software that performs some function at run-time.
 - Examples: programs, objects, processes, clients, servers, databases

Connectors

- A connector is a mechanism that mediates communication, coordination, or cooperation among components.
- Connectors describe the interactions among components.
- Implementations of connectors are usually distributed over many system components; often do not correspond to discrete elements of the running systems.
 - Examples: shared variable access, procedure calls, remote procedure calls, communication protocols, data streams, transaction streams.

Definition of Software Architecture

- **Software Architecture** is an abstract representation, or model, of a software system in terms of a structure that consists of a collection of elements together with the relationships among them to achieve software design purposes and to manifest a certain set of design properties of the system.

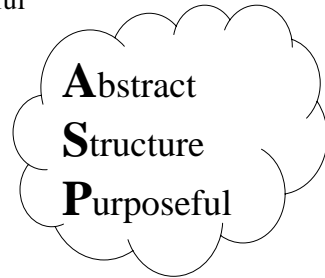
U08182 © Peter Lo 2010

4

- The details of the elements and relationships are hidden and replaced with abstract computational entities, called components and connectors, respectively.
- These abstract entities are either represented by a number of characteristic properties that affect the properties of the system or a composition of such abstract entities of lower level in the form of an architectural model.

Common Features in Architecture Notion

- Architecture is Abstract
- Architecture is about Structure
- Architecture is Purposeful



U08182 © Peter Lo 2010

5

Architecture is Abstract

- Software architecture is an abstract representation of a complicated software system.

Architecture is about Structure

- Software architecture represents a software system in terms of the components together with the interactions among them.
- The details of the components and connectors are hidden and replaced with abstract computational entities that are characterised by a set of properties or architectural models at lower level.

Architecture is Purposeful

- Software architecture is a model of software systems built to achieve certain software engineering purposes, which include as design documentation, as transferable knowledge about software design, etc.

The Importance of Architecture

- Communication among stakeholders
- Manifestation of early design decisions
- Representation of transferable knowledge of a system

U08182 © Peter Lo 2010

6


- Communication among stakeholders
- Manifestation of early design decisions
 - It defines constraints on its implementation
 - It dictates organizational structure
 - It inhibits or enables a system's quality attributes
 - It's the earliest point at which the system can be analysed
 - It enables predictions on system's quality
 - It makes reasoning about and managing changes easier.
 - It helps in evolutionary prototyping
- Representation of transferable knowledge of a system


Software Architecture Visual Notation


- Architectural Element (i.e. Components) are represented as **Node** and relationships (i.e. Connectors) between the architectural elements are represented as **Arrows** between nodes.

Software Architecture Visual Notation – Component

- Components in the architecture are represented as Nodes
 - ◆ **Hardware Component** – represented by square corner rectangles
 - ◆ **Passive Software Component** – represented by round corner rectangles for software components that are not active (i.e. procedure)
 - ◆ **Active software Component** – represented by rectangles with corners cut off (i.e. run-time processes)

 Hardware component

 Passive software component

 Active software component

- **Square corner rectangles:** they represent hardware component
- **Round corner rectangles:** they represent software components that are not active
- **Rectangles with corners cut off:** they represent active software components, i.e. run-time processes

For example, a procedure is a software component. It is passive because it only executes when it is called. Therefore, in the visual notation a procedure is represented by a round corner rectangle. A process is a run-time entity. It is active and can execute concurrently with other processes. Therefore, in the visual notation, it is presented by rectangle with corners cut off.

Software Architecture Visual Notation – Control

- Control is indicated via **Solid Lines**
- A component that has a control feature if it is executable.
- A connection has a control feature, if the connection can pass the control from one component to another and causes its execution.

—— Control

Software Architecture Visual Notation – Data

- Data are indicated via **Dotted Lines**.
- A component that has a data feature means if the component stores data as its internal state.
- A connection has a data feature, if data is passed from one component to another.



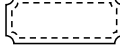


..... Data

Software Architecture Visual Notation – Relationships/Connectors

- Relationships between architectural elements are represented as **Arrows** between the nodes.
- A connection has a control feature, if the connection can pass the control from one component to another and causes its execution. A connection has a data feature, if data is passed from one component to another.

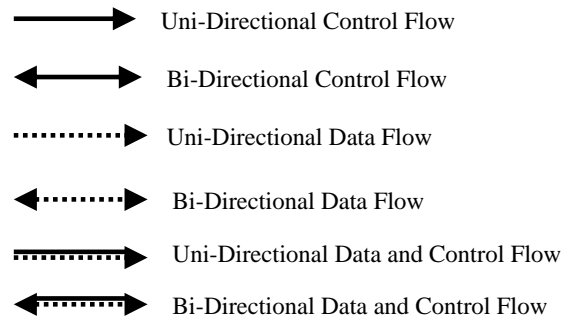
→ Relationship

Summary of Visual Notation – Software Components

	Processes without internal state
	Objects, abstract data types, modules
	Databases, processes with internal states
	Files, variables, constants
	Functions / procedures without side effects

- A component that has a control feature means that it is executable. A component that has a “Data” feature means that the component stores data as its internal state. If it is also executable, the component retains the states between executions of the component. A typical example of such computation component is object.
- For example, if a computation component is depicted as a rounded rectangle with a pair of lines, one solid and one dotted, the component must have both data and control features. That is, it must be executable and it also contains storage of data to store its state. By contrast, a passive data component that is not executable, such as a text file, is described by a single dotted line indicating that a file only contains data.

Summary of Visual Notation – Connections



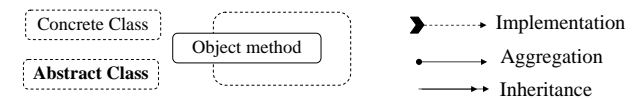
U08182 © Peter Lo 2010

13

- For example, data Connection (such as socket connections) are shown as dotted lines, whereas control connections (such as process spawning) are shown as solid line.
- A procedure call that passes data from the caller to a procedure is depicted as an arrow consisting of two lines, one solid line and one dashed line. The solid line indicates that the procedure call causes the control transmitted to the procedure. The dashed line indicates that data are also passed from the caller through parameters to the procedure.

Summary of Visual Notation – Additional Notation for OO

- A class (or an object) may have a number of methods that can be call by other components.
- These methods are drawn on the boarder of the class to indicate that they are visible from the outside of the element.



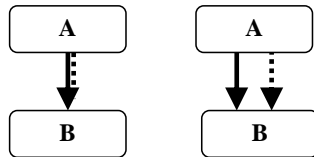
U08182 © Peter Lo 2010

14

- In addition to arrows for connections, there are also arrows for relationships between architectural elements of different views.
- For example, there are arrows for one element which implements another, arrows for one element which is an aggregate of a number of others, and an element that inherits the features of another.

Data Flow and Control Flow between Components

- Having a data and control flow between two components is different from having a data flow and a control flow between them.
- The former shows one connector between two components that data and control are passed from component A to B at the same time. The latter shows two connectors between the components, which means that control and data can be passed separately (e.g. at different time).



U08182 © Peter Lo 2010

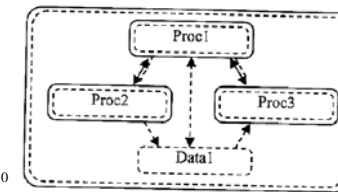
15

Note on Control and Data

- A component may have both data and control features, if it is executable and also retains the states between executions of the component.
 - A typical example of such computation component is object.
- A connection may also have both control and data features.
 - A typical example of such connector is a procedure call that contains parameters.

Composition of Architectural Elements

- When an architectural element is a composition of a number of lower level elements, the node can be extended into a box so that the lower level elements and their relationships can be drawn inside the box. In such case, the node looks like a box which bind the elements together.

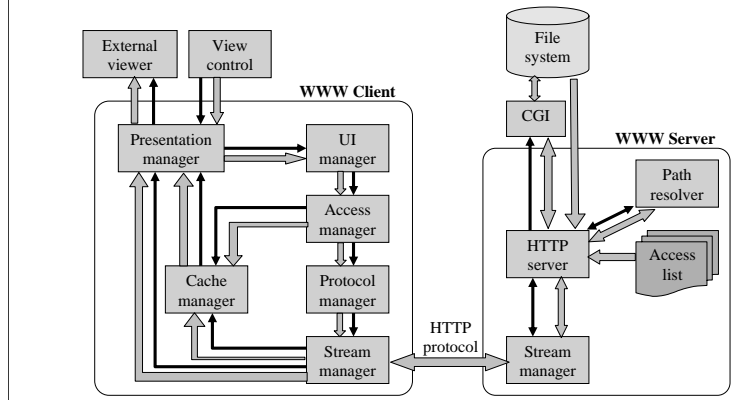


U08182 © Peter Lo 2010

16

The above diagram shows a procedure that has three procedure and one local passive data elements (i.e. a local variable) as its components and the relationships between these elements. As shown in the diagram, Proc1 call Proc2 and Proc3, and passes data through parameters to the procedures. Procedure Proc1 also writes data into and read data from the local variable Data1. Procedure Proc2 only writes into Data1. Procedure Proc3 only read from Data1.

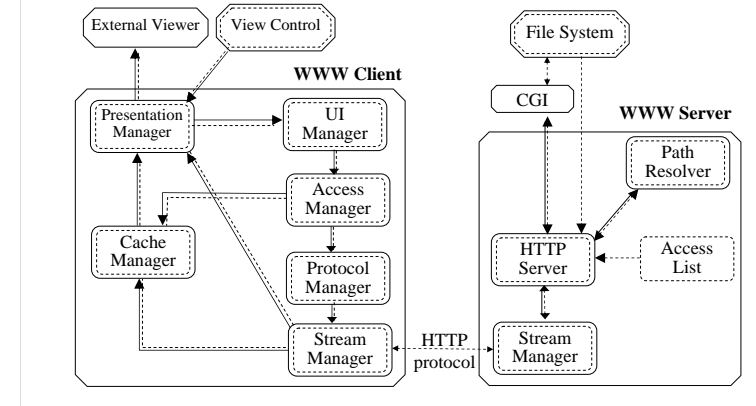
Example 1: Description of WWW Client-Server Architecture



At the top level, there are 6 components in this architecture: the external viewer, the view control, the Client side (WWW browser), the web server, the common gateway interface (CGI), and a file system.

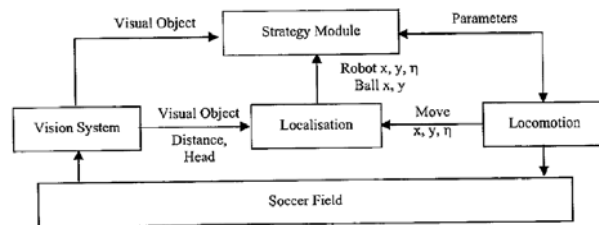
- An external viewer is a program that is used to view certain types of contents of a web page, such as a postscript viewer, QuickTime for playing video and audio stream, etc. They are active run-time processes.
- The view control component maintains a configuration file and aids the mapping of document types to external viewers when consulted by the presentation manager of the browser.
- On the server side, the file system maintains a collection of files in HTML format and other data as well. These files are retrieved by the WWW server when requested by the browser.
- CGI executes the scripts embedded in HTML files and provides extended functionality of the system for implementation of web-based applications.
- The WWW browser executes on the client side. It can be decomposed into a number of components, which include:
 - User Interface (UI) manager handles the look and feel of client's user interface.
 - Presentation manager delegates information display to UI manager or external program (external viewers) to view resources that are not directly supported by the user interface manager.
 - Access manager accepts the information requests in the form of URLs captured by the UI manager and determines if the request URL exists in the Cache and also interprets history based navigation, such as "Back".
 - Cache manager a collection of retrieved files and passes a file to the presentation manager if it is requested by the access manager.
 - Protocol manager determines the types of request for information captured by access manager and invokes the appropriate protocol suite to service the request.
 - Stream manager uses the protocol invoked by protocol manager to communicate with the server in order to obtain the requested information.
- On the WWW Server side, there are also a number of components, which include:
 - Stream manager communicates with the stream manager on the client side to receive information request and send back the requested information.
 - Access list stores a list of clients that are authorized for the documents.
 - Path resolver resolves the location of the requested documents in the files.
 - The HTTP server ensures transparent access to the file system where the source documents are stored. It also consults an access list to determine if the requesting client is authorized to access the data pointed to by the URL. CGI scripts are passed to CGI to execute.

Example 1: Description of WWW Client-Server Structure in Visual Notation



Example 2: The Software Structure of Robot Soccer UNSW

- The software consists of 4 components. These components are all active processes. The connections between the components only pass data between them.



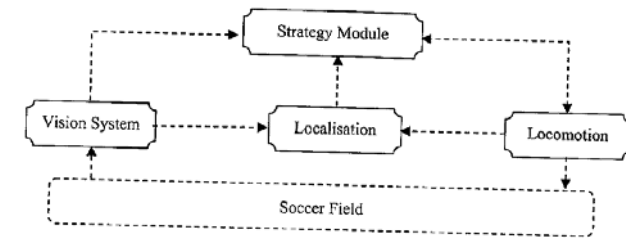
U08182 © Peter Lo 2010

19

UNSW's software consists of the following four components to implement the basic skills such as dribbling, head butting and kicking, as well as game strategies.

- **Vision:** The images captured by the robot's camera are processed to recognize blob, converting them into objects such as beacons, goals, or the ball. Metrics such as direction and distance are generated at the same time.
- **Localization:** It maintains three variables: the x and y coordinates and the heading of the robots. It updates the position and direction of the robot each time field objects are recognized and each time the robot moves.
- **Locomotion:** It drives the legs and the head effectors based on directions from the strategy module.
- **Strategy modules:** It combines various skills and behaviors to achieve the goal of winning the matches.

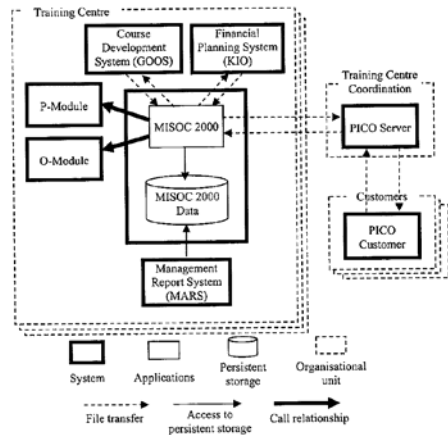
Example 2: The Architecture of UNSW in Visual Notation



U08182 © Peter Lo 2010

20

Example 3: Computer systems in Dutch Army Training Centers

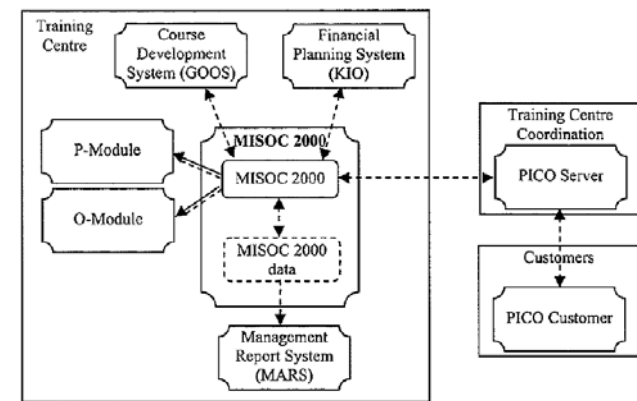


21

Each training centre has a number of computer system interconnected together. Their functions and interrelationships are described below:

- The *course development system (GOOS)* is used for developing new courses. It used information from MISOC 2000, such as the number of registrations for a course and the availability of locations. After new courses have been developed with GOOS, they are imported into MISOC 2000. From then on it is possible to register students for these courses. The data exchanges between MISOC 2000 and GOOS consist of files being imported and exported.
- The *financial planning system (KIO)* is used for calculation of the costs. KIO feeds MISOC 2000 with information concerning cost centers and retrieves information from MISOC 2000 concerning the organization, instructors, locations, and resources. These data exchanges take the form of file transfers.
- The *management reporting system (MARS)* is a management information system that is used for generating various management reports. This system is implemented using a COTS report tool. This tool directly accesses the MISOC 2000 database to retrieve information.
- The *P-module* is a part of the *human resource (HR)* information system. The HR system store information about employees, such as name, rank and qualifications. This information is maintained both at the central level for the whole DoD, and at the local level for each unit. At the local level, each unit uses an instance of the P-module for managing the information of the employees of that unit. Periodically, the central system feeds the P-module of each unit with information concerning the employees of that unit using file transfer. These downloads are one way only, so global updates to the human resources information are only possible at the central HR system. However, the P-module does provide facilities for performing updates, but these changes are not carried through to other units. This enables units to register temporary staff. The P-module plays two different roles: (a) as a stand-alone system for managing personnel information and (b) as a "service" for other system to access personnel information. MISOC 2000 uses the P-module in the latter roles, mostly to retrieve information about instructors. When MISOC 2000 invokes the P-module, one of the applications of the P-module is started on the user's workstation and control is transferred to that application. After the user has performed the necessary actions, the application is closed and control is returned to MISOC. Other system use P-module in similar ways.
- The function of the *O-module* is to provide access to information concerning the organization and its resources. Just like the human resource information, this information is stored at both the central and the local level and the central mainframe performs periodical downloads to local instances of the O-module.
- MISOC 2000 is also related to PICO (Planning and Development system for Course and Training) system outside the training centers. PICO gathers the course information of all training centers and enables their customers, i.e. all organizational units, to enroll employees for these courses. The information concerning the courses is transferred from the training centers to a central server, the PICO server. The customers of the training centers use a local system, "PICO Customer", to retrieve this information and enroll their employees for these courses. Finally these enrollments are transferred from the PICO server to MISOC 2000 at the appropriate training centre. All of these flows of information are file transfers.

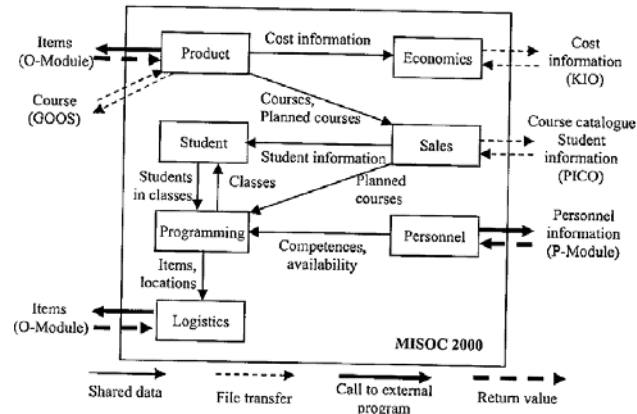
Example 3: Representation of the Information System in Visual Notation



U08182 © Peter Lo 2010

22

Example 4: The Structure of MISOC 2000



U08182 © Peter Lo 2010

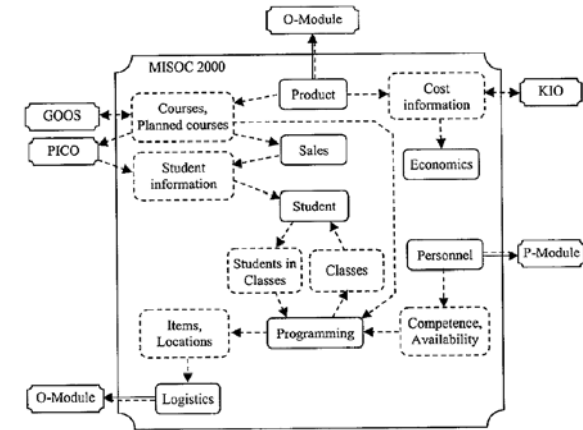
23

MISOC 2000 consists of a number of sub-systems. The choice of sub-systems in the design of MISOC 2000 was driven by the processes of the training centers: each sub-system supports a specific group of users. The following sub-systems were recognized.

- *Product*: formulating course catalogues and production plans for a training centre
- *Sales*: distribution of course catalogues and recording agreements with customers
- *Students*: registration of student information
- *Programming*: creating short-term schedules
- *Logistics*: management of the availability of locations and items
- *Economics*: exporting cost information to KIO and importing information about cost centers from KIO
- *Personnel*: an extension of the P-module to record personnel information specific for training centers

These sub-systems communicate through a shared database. The information that share and their communication with the systems in the environment.

Example 4: Representation of MISOC 2000's architecture in Visual Notation



24

What is in a Style?

- A set of component types
- A topological structure
- A set of semantic constraints
- A set of connectors

U08182 © Peter Lo 2010

25

A set of component types

- A pipe-and-filter system consists of a number of filters
- A data abstraction architecture consists of a number of instances of abstract data types and/or objects
- Other common types of components: processes, procedures, files

A topological structure

- A software architecture also limits the choices of the topological structure that components are interconnected through connectors.

A set of semantic constraints

- The state of an object in an data abstraction architecture can only be changed through calls to its methods
- The filters in a pipe-and-filter architecture do not share states with each other
- One layer can only call the service provided by the lower layer in a layered system

A set of connectors

- Examples of connectors: subroutine calls, remote procedure calls, data streams, and sockets

What is Ambiguous in a Style?

- The number of components involved
- The mechanism of interaction
- The function of the system and components

U08182 © Peter Lo 2010

26

The number of components involved

- For example, a pipe-and-filter architecture may have two filters connected together by a pipe or 20 filters connected by 19 pipes.

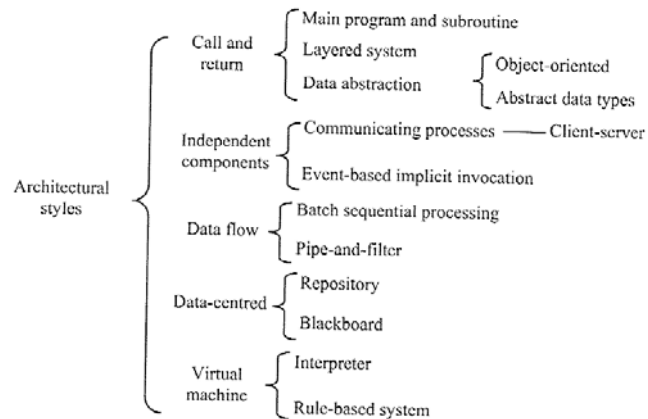
The mechanism of interaction

- For example, in a layered system, the call to the lower layer can be local procedure calls, or remote procedure calls, or other process communication mechanisms.

The function of the system and components

- For example, one of the components in an architectural style may be a database, but the kind of data may vary.

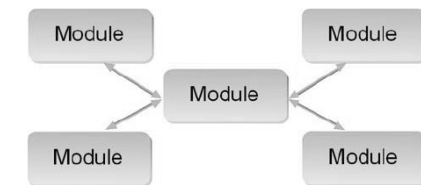
Software Architectural Styles



- One of the hallmarks of software architectural design is the use of idiomatic patterns of system organisation.
- An architectural style defines a family of systems in terms of a pattern of structural organisation.
 - A vocabulary of component and connector types
 - A set of constraints on how they can be combined
 - One or more semantic models that specify how to determine a system's overall properties from the properties of its parts

Call-and-Return Architectures

- Call-and-Return Architecture has been the dominant architectural style in large system for the past 30 years.
- A number of sub-types of the style have emerged including



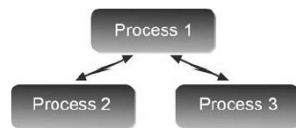
U08182 © Peter Lo 2010

28

Most systems built today use some form of a Call-and-Return architecture pattern. This pattern includes object-oriented architectures, remote procedure call (RPC) architectures, and tiered or layered architectures. The primary goal of these approaches is to easily accommodate changes in scale and feature function. For example, one can employ more than one processor using RPC to increase performance, or add new feature functionality using the layered approach.

Independent Component

- The architectural style of independent components has attracted increasing interest recently for its strong support to software reuse and evolution due to its ease of integration of components into a system



U08182 © Peter Lo 2010

29

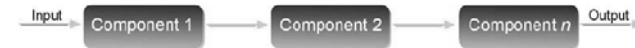
Independent Component Architecture patterns are also known as Messaging Architectures. Each component in this architecture communicates with other components through messages. Thus, one component in a system does not have direct control of other components. The messages may be sent to a specific component or broadcast to all interested components through a publish/subscribe mechanism, but there is no guarantee when the components will process the received message.

Typically, event driven systems implement a publish/subscribe mechanism. Publisher components announce what data they are going to broadcast. Subscriber components then register with the publisher for the information they are interested in receiving. A message manager acts as a communication broker between the publisher and subscriber components.

The primary advantage of the Independent Component Architecture is that each component is completely decoupled from the others. This allows the components to run in parallel, facilitating higher scalability and performance.

Data Flow

- A data flow system is one in which
 - ◆ The availability of data controls the computation
 - ◆ The structure of the design is dominated by orderly motion of data from process to process
 - ◆ The pattern of data flow is explicit
- It is widely used in various application domains where data processing plays a significant role.
- In a pure data flow system, there is no other interaction between processes.



U08182 © Peter Lo 2010

30

Data-flow Architecture patterns promote reuse of components and are built to accommodate change in feature function. A system built in this pattern will be composed of individual components that operate on data input in series and in succession. Input to the system flows through each component individually until the system is complete. Examples of this architecture are a series of XSL transforms, and the classic pipe-and-filter mechanism made famous by the inventors of UNIX.

A great advantage of using data-flow architecture is its simplicity. There are no complex components to manage, and any combination of the components can produce a different system. However, due to the way a problem is dissected into individual components, the system is inherently single threaded, making it difficult to increase performance.

Data-Centered

- The term data-centered architecture refers to systems in which the accesses and update of a widely accessed data-store is an apt description
- The data-centered style offers a structural solution to integrability especially when building from existing systems.



U08182 © Peter Lo 2010

31

It is often difficult to assimilate data from multiple systems into one view. Data-centered patterns are useful when data integration is important (i.e. Data warehousing). In this architecture, clients and the data store are independent of each other. Thus, it is easy to add new clients, or change views of the data and make these changes available to all clients.

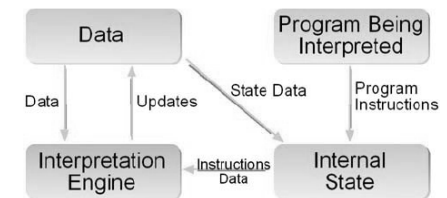
The primary purpose of a Data-centered Architecture pattern is to provide access to a heavily used data store. Clients of the data store are notified of changes in one of two ways: the client is directly notified of changes, or the client queries the data store for changes. A data store that notifies clients of data changes is called an active repository. An active repository broadcasts changes to any client that has subscribed for updates. This model is

also called a publish/subscribe model, since the repository acts as the publisher and the clients subscribe to their desired updates.

In a passive repository model, data is simply updated as requested. The clients must request any changes that have been made to the data store. This is the model that typical database applications follow.

Virtual Machine

- A software system of virtual machine architecture usually consists of four components interconnected
 - ◆ Data (Program State)
 - ◆ Program being Interpreted
 - ◆ Interpretation Engine
 - ◆ Internal State



U08182 © Peter Lo 2010

The primary purpose of a Virtual Machine (VM) Architecture pattern is to make a software system portable. The Java Runtime Environment (JRE) and the Common Language Runtime (CLR) are good examples of virtual machines that can handle portability issues.

Another use for virtual machines is simulation. A VM can simulate a hardware platform or even a system that has not been built. Most developers who build software for cell phones, PDA, and similar devices use a hardware simulator to test their system before loading the system onto the device. The primary drawback to running a program through a virtual machine is the performance cost resulting from the extra work required to run the program and the simulation code.

Portability is the main advantage. For example, the Java language is built to run on top of the Java Virtual Machine, which allows the language to be platform independent.

Combinations of Styles

- The design of a software system may need to combine different styles to solve the design problem.
- Heterogeneous style
 - ◆ Systems that are not in a single style
- How styles can be combined together?
 - ◆ Hierarchically Heterogeneous Styles
 - ◆ Simultaneously Heterogeneous Styles
 - ◆ Locationally Heterogeneous Styles

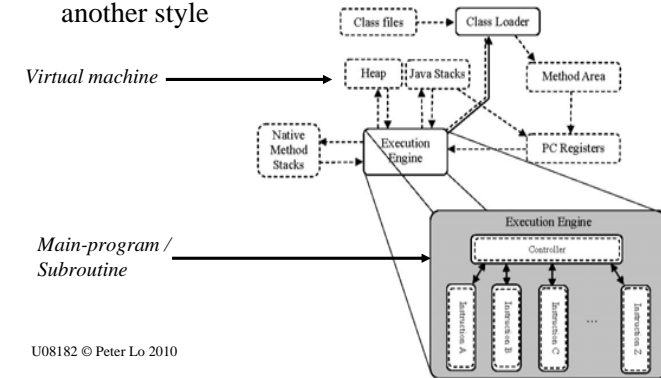
U08182 © Peter Lo 2010

33

Most software systems are not built from a single style. Instead, the design of a software system at architectural level often need to combine different styles to solve the design problem. Software systems that are not in a single style are called Heterogeneous style.

Hierarchical Heterogeneous Styles

- Whole system in a style, while a component in another style



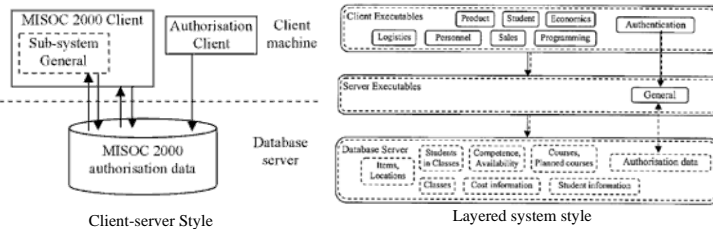
U08182 © Peter Lo 2010

The hierarchical combination of different styles in the design of a software system is to use one architectural style at one level of abstraction while using a different style in the design of a component of the higher level.

For example, at the top level of abstraction we can use virtual machine style to design the Java virtual machine (JVM). The key component of the architecture is the execution engine. We can use the main-program-and-subroutine style to implement this component. The component consists of a controller that decides which instruction to be executed while each instruction is implemented by a procedure.

Simultaneously Heterogeneous Styles

- The architecture of a system can be described as a number of different styles



U08182 © Peter Lo 2010

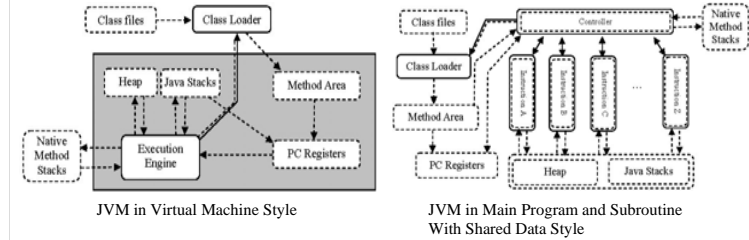
35

In many cases, the architecture of a software system can be aptly described as in a number of different architectural styles.

An concrete example of system in simultaneously heterogeneous style is the MISOC 2000 system. It can be viewed as in the client-server style. It can also be viewed as a layered system, where the clients call the server via remote procedural calls. Moreover, the system is implemented in an object-oriented programming language. We can also view the system from an object-oriented style point of view.

Locationally Heterogeneous Styles

- Different subsets of the components and connectors fall into different architectural styles



U08182 © Peter Lo 2010

36

A software system's architecture is in a locationally heterogeneous style if it can be considered as in one style when taking a subset of its components and connectors, meanwhile it can also be viewed as in another style when taking a different subset of the components and connectors.

For example, if we redraw the architecture diagram of the Java virtual machine, we can find that it is in the Main Program and Subroutine With Shared Data Style.

Further Readings

- Zhu, H., Software Design methodology. Chapter 4, 5, 6, pp73-169.
- Shaw, M. and Garlan, D., Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 1996. Chapter 2: Architectural Styles, pp19~32.
- Bass, L., Clements, P. and Kazman, R., Software Architecture in Practice, Addison Wesley, 1998.
 - ◆ Chapter 2: What is Software Architecture? pp21~44.
 - ◆ Chapter 4: Quality Attributes, pp75~92.
 - ◆ Chapter 5: Moving From Qualities to Architecture: Architectural Styles, pp93~122

U08182 © Peter Lo 2010

37

Website about software architecture at Software Engineering Institute of Carnegie Mellon University:

<http://www.sei.cmu.edu/architecture/definitions.html>

An Introduction to Software Architecture

http://www.cs.cmu.edu/afs/cs/project/vit/ftp/pdf/intro_softarch.pdf

Software Architecture: An Overview

<http://www.sds-consulting.com/SoftwareArchitectureOverview09.pdf>