SENG 401
Software Architecture

Lecture 8
Topic – Documenting Software Architecture
Sub-Topic – Component and Connector Styles

Gias Uddin
Assistant Professor, University of Calgary

http://giasuddin.ca/

Books Used/Cited in this Course

ID	Title
B1	Software Architecture: Foundations, Theory, and Practice. Taylor, Medvidovic, Dashofy. Wiley, 2009. *Available at UofC Library*
B2	The Architecture of Open-Source Applications. Amy Brown, Greg Wilson. Volume I. http://www.aosabook.org/en/index.html
B3	The Architecture of Open-Source Applications. Amy Brown, Greg Wilson. Volume II. http://www.aosabook.org/en/index.html
B4	**Documenting Software Architecture: Views and Beyond, 2nd Edition. Addison-Wesley, 2010. Clements et al.** ***Available at UofC Library***
B5	Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, 2nd Edition. Nick Rozanski. Addison-Wesley Professions, 2011. *Available at UofC Library*
B6	Software Architecture in Practice. Bass, Clements, Kazman. Addison-Wesley, 2012 *Available at UofC Library*

SCHULICH
School of Engineering

Documenting Software Architecture
Topic: Collection of Software Architectural Styles
Sub-Topic: Component and Connector Styles

References: B4 – C4

C&C Styles

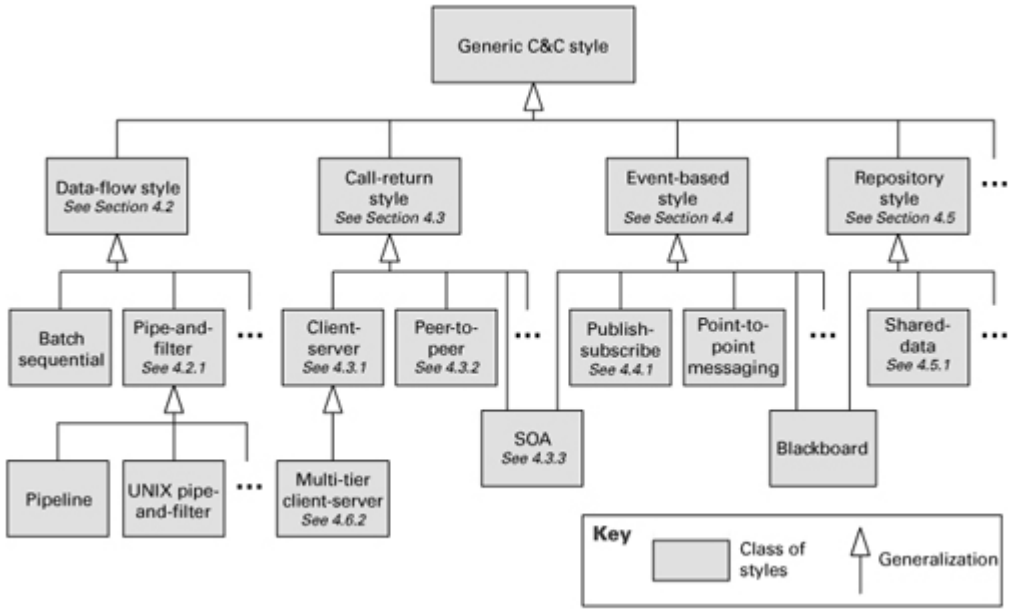
4 Major Types

A component-and-connector (C&C) style introduces a specific set of component-and-connector types and specifies rules about how elements of those types can be combined. Additionally, given that C&C views capture runtime aspects of a system, a C&C style is typically also associated with a computational model that prescribes how data and control flow through systems designed in that style.

Style Name	What is it?
1. Data flow styles	Styles in which computation is driven by the flow of data through the system.
2. Call-return styles	Styles in which components interact through synchronous invocation of capabilities provided by other components.
3. Event-based styles	Styles in which components interact through asynchronous events or messages.
4. Repository styles	Styles in which components interact through large collections of persistent, shared data.

C&C Styles

A partial representation of the space of C&C styles



Naturally this is only a partial representation of the space of C&C styles: there are other general categories, and there are many styles that are specializations of these categories. Additionally, in most real systems several styles may be used together, often from across categories. For example, enterprise IT applications are frequently a combination of client-server and shared-data styles.

2. Call-Return Styles

Service-Oriented Architecture (SOA) Styles

SCHULICH School of Engineering

Service-oriented architectures consist of a collection of distributed components that provide and/or consume services. In SOA, service provider components and service consumer components can use different implementation languages and platforms. Services are largely standalone: service providers and service consumers are usually deployed independently, and often belong to different systems or even different organizations.

Elements	• Service providers, which provide one or more services through published interfaces. Properties will vary with the implementation technology (e.g., EJB or ASP.NET) but may include performance, authorization constraints, availability, and cost. These properties are specified in Service-Level-Agreement (SLA). • Service consumers, which invoke services directly or through an intermediary. • ESB, which is an intermediary element that can route and transform messages between service providers and consumers • Registry of services, which may be used by providers to register their services and by consumers to query and discover services at runtime. • Orchestration server, which coordinates the interactions between service consumers and providers based on scripts that define business workflows. • SOAP connector, which uses SOAP (Simple Object Access Protocol) for synchronous communication between web services, typically over HTTP. Ports of components that use SOAP are often described by WSDL (Web Services Description Language) • REST (Representational State Transfer) connector, which relies on the basic request/reply operations of the HTTP protocol • Messaging connector, which uses a messaging system to offer point-to-point or publish-subscribe asynchronous message exchange.

2. Call-Return Styles

Service-Oriented Architecture (SOA) Styles

Relations	Attachments of the different kinds of ports available to the respective connectors
Computational Model	Computation is achieved by a set of cooperating components that provide and/or consume services over a network. The computation is often described as a kind of workflow model.
Constraints	• Service consumers are connected to service providers, but intermediary constraints (e.g., ESB, registry, or BPEL server) may be used. ESB is an Enterprise Service Bus is a connector that facilitates communication between components in an SOA. BPEL is Business Process Execution Language is an executable language to specify actions within business processes with web services. • ESBs lead to hub-and-spoke topology • Service providers may also be the service consumers • Specific SOA patterns impose additional constraints
What it's for	• The main benefit is interoperability. Because service providers and service consumers may run on different platforms, service-oriented architectures often integrate different systems and legacy systems. Service-oriented architecture also offers the necessary elements to interact with external services available over the Internet. • Special SOA components such as the registry or the ESB also allow dynamic reconfiguration, which is useful when there's a need to replace or add versions of components with no system interruption.

2. Call-Return Styles

Service Mediation in SOA

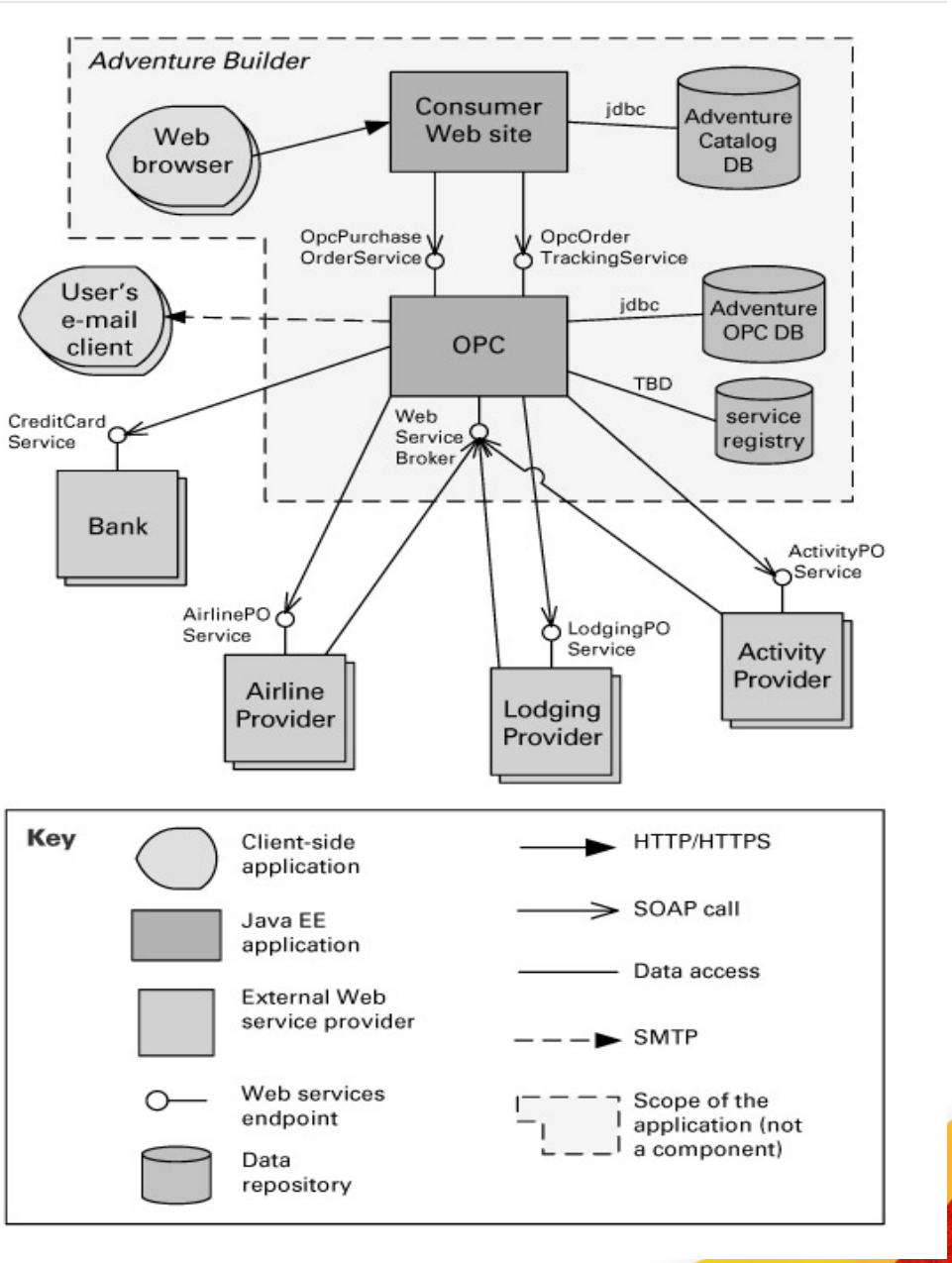
In addition to the service provider and consumer components that you develop, an SOA application may use specialized components that act as intermediaries and provide infrastructure services:	
ESB	Service invocation can be mediated by an enterprise service bus (ESB). An ESB routes messages between service consumers and service providers. In addition, an ESB can convert messages from one protocol or technology to another, perform various data transformations (for example, format, content, splitting, merging), perform security checks, and manage transactions. When an ESB is in place, the architecture follows a hub-and-spoke design, and interoperability, security, and modifiability are improved. When an ESB is not in place, service providers and consumers communicate to each other in a direct point-to-point fashion.
Service Registry	To improve the transparency of location of service providers, a service registry can be used in SOA architectures. The registry is a component that allows services to be registered and then queried at runtime. It increases modifiability by making the location of the service provider transparent to consumers and permitting multiple live versions of the same service.
Orchestration Server	An orchestration server (or orchestration engine) is a special component that executes scripts upon the occurrence of a specific event (for example, a purchase order request arrived). It orchestrates the interaction among various service consumers and providers in an SOA system. Applications with well-defined business workflows that involve interactions with distributed components or systems gain in modifiability, interoperability, and reliability by using an orchestration server. Many orchestration servers support the Business Process Execution Language (BPEL) standard.

2. Call-Return Styles
Connectors in SOA

Type	Description
Call-Return Connectors	• SOAP is the standard protocol for communication in Web services technology. Service consumers and providers interact by exchanging request/reply XML messages, typically on top of HTTP. • With the REST connector, a service consumer sends synchronous HTTP requests. These requests rely on the four basic HTTP commands (post, get, put, and delete) to tell the service provider to create, retrieve, update, or delete a resource (a piece of data). Resources have a well-defined representation in XML, JSON, or a similar language/notation.
Asynchronous Messaging	Components exchange asynchronous messages, usually through a messaging system such as IBM WebSphere MQ, Microsoft MSMQ, or Apache ActiveMQ. The messaging connector can be point-to-point or publish-subscribe. Messaging communication typically offers great reliability and scalability.
Service Connector Interface	Components have interfaces that describe the services they request from other components and the services they provide. Components initiate actions to achieve their computation by cooperating with their peers by requesting services from one another.
In practice, SOA environments may involve a mix of the three connectors listed above, along with legacy protocols and other communication alternatives (such as SMTP).	

2. Call-Return Styles
Example of an SOA

Diagram of the SOA view for the Adventure Builder system. The OPC (Order Processing Center) component coordinates the interaction with internal and external service consumers and providers. This system interacts via SOAP Web services with several other external service providers. Note that the external providers can be mainframe systems, Java systems, or .NET systems—the nature of these external components is transparent because the SOAP connector provides the necessary interoperability.



3. Event-Based Style

- Event-based styles allow components to communicate through asynchronous messages. Such systems are often organized as a loosely coupled federation of components that trigger behavior in other components through events.
- A variety of event styles exist. In some event styles, connectors are point-to-point, conveying messages in a way similar to call-return, but allowing more concurrency, because the event sender need not block while the event is processed by the receiver.
- In other event styles, connectors are multi-party, allowing an event to be sent to multiple components. Such systems are often called publish-subscribe systems, where the event announcer is viewed as publishing the event that is subscribed to by its receivers.

3. Event-Based Style
Publish-Subscribe Style

- Components interact via announced events. Components may subscribe to a set of events. It is the job of the publish-subscribe runtime infrastructure to make sure that each published event is delivered to all subscribers of that event. Thus, the main form of connector in this style is a kind of event bus. Components place events on the bus by announcing them; the connector then delivers those events to the components that have registered an interest in those events.
- The publish-subscribe style can take several forms. In one common form, called implicit invocation, the components have procedural interfaces, and a component registers for an event by associating one of its procedures with each subscribed type of event. When an event is announced, the associated procedures of the subscribed components are invoked in an order usually determined by the runtime infrastructure. Graphical user-interface frameworks, such as Visual Basic, are often driven by implicit invocation: User code fragments are associated with predefined events, such as mouse clicks.
- In another publish-subscribe form, events are simply routed to the appropriate components. It is the component's job to figure out how to handle the event. Such systems put more of a burden on individual components to manage event streams, but also permit a more heterogeneous mix of components than implicit invocation systems do.
- In some publish-subscribe systems, an event announcer may block until an event has been fully processed by the system. For example, some user-interface frameworks require that all views be updated when the data they depict has been changed. This is accomplished by forcing the component that announces a "changed-data" event to block until all subscribing views have been notified.

3. Event-Based Style

Publish-Subscribe Style

SCHULICH

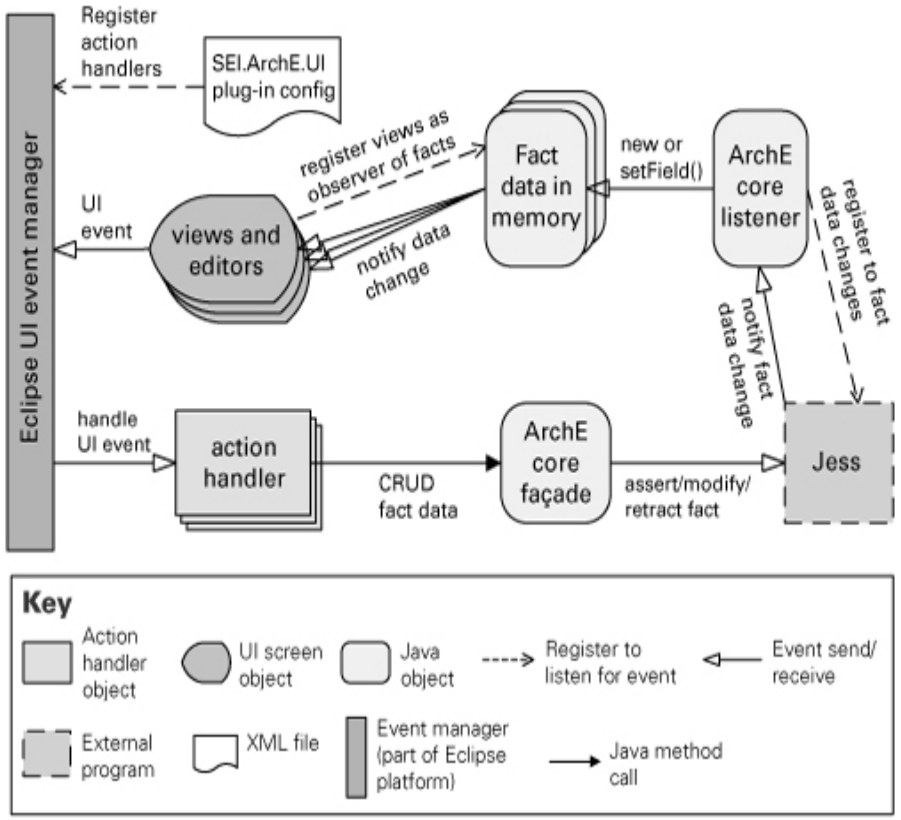
Elements	• Any C&C component with at least one publish/subscribe port. Properties vary, but they should include which events are announced and/or subscribed to, and the conditions under which an announcer is blocked. • Publish-subscribe connector, which will have two roles (announce and listen) for components that wish to publish and/or subscribe to events.
Relations	Attachment relation associates components wit the publish-subscribe connector by prescribing which components announce events and which components have registered to receive events
Computational Model	Components subscribe to events. When an event is announced by a component, the connector
Constraints	All components are connected to an event distributor that may be viewed as either a bus – that is, a connector – or a component. Publish posts are attached to announce roles, and subscribe ports are attached to listen roles. Constraints may restrict which components can listen to which events, whether a component can listen to its own events, and how many publish-subscribe connectors can exist within a system. A component may be both a publisher and a subscriber, by having ports of both types.
What it's for	• Sending events to unknown recipients, isolating event producers from event consumers • Providing core functionality for GUI frameworks, mailing lists, bulletin boards, and social networks.
Relation to other styles	• The publish-subscribe style is similar to a blackboard repository style, because in both styles components are automatically triggered by changes to some component. However, in a blackboard system, the database is the only component that generates such events; in a publish-subscribe system, any component may generate events. • Implicit invocation is often combined with call-return in systems in which components may interact either synchronously by service invocation or asynchronously by announcing events.

3. Event-Based Style

Example of Publish-Subscribe Style

publish-subscribe view of the SEI ArchE tool. There are three different publish-subscribe interactions in this architecture:

Type	Description
UI events	Eclipse UI event manager acts as an event bus for user-interface events (such as button clicks). Subscription information—that is, what UI events are relevant to the system and what components handle them—is defined at load time when the event manager reads the SEI.ArchE.UI plug-in config XML file. From then on a UI event generated by the user working on a view or editor is dispatched via implicit invocation to the action handler objects that subscribe to that event.
Data events	The data manipulated in ArchE is stored using a rule engine called Jess. Data elements are called facts. When a user action creates, updates, or deletes a fact, that action generates respectively an assert, modify, or retract fact event that is sent to Jess. When Jess processes that event, changes to many other facts may be triggered. Jess also acts as an event bus that announces changes to facts. In the ArchE architecture, there is one component that subscribes to all data changes: ArchE core listener.
Observers	ArchE keeps in memory copies of the fact data elements persisted in the rule engine. These copies are observable Java objects. User-interface screens (that is, views) that display those elements are observers of the fact data objects. When facts in memory are created or updated, the views are notified.



4. Repository Styles

Overview

- Repository views contain one or more components, called repositories, which typically retain large collections of persistent data. Other components read and write data to the repositories. In many cases access to a repository is mediated by software called a database management system (DBMS) that provides a call-return interface for data retrieval and manipulation. MySQL is an example of a DBMS. Typically a DBMS also provides numerous data management services, such as support for atomic transactions, security, concurrency control, and data integrity. In C&C architectures where a DBMS is used, a repository component often represents the combination of the DBMS program and the data repository.
- Repository systems where the data accessors are responsible for initiating the interaction with the repository are said to follow the shared-data style. In other repository systems, the repository may take responsibility for notifying other components when data has changed in certain prescribed ways. These systems follow the blackboard style. Many database management systems support a triggering mechanism activated when data is added, removed, or changed. You can employ this feature to create an application following the blackboard style. But if your application uses the DBMS for retrieving and changing data in the repository but doesn't employ triggers, you're following the pure shared-data style.

4. Repository Styles

Shared-Data Style

SCHULICH School of Engineering

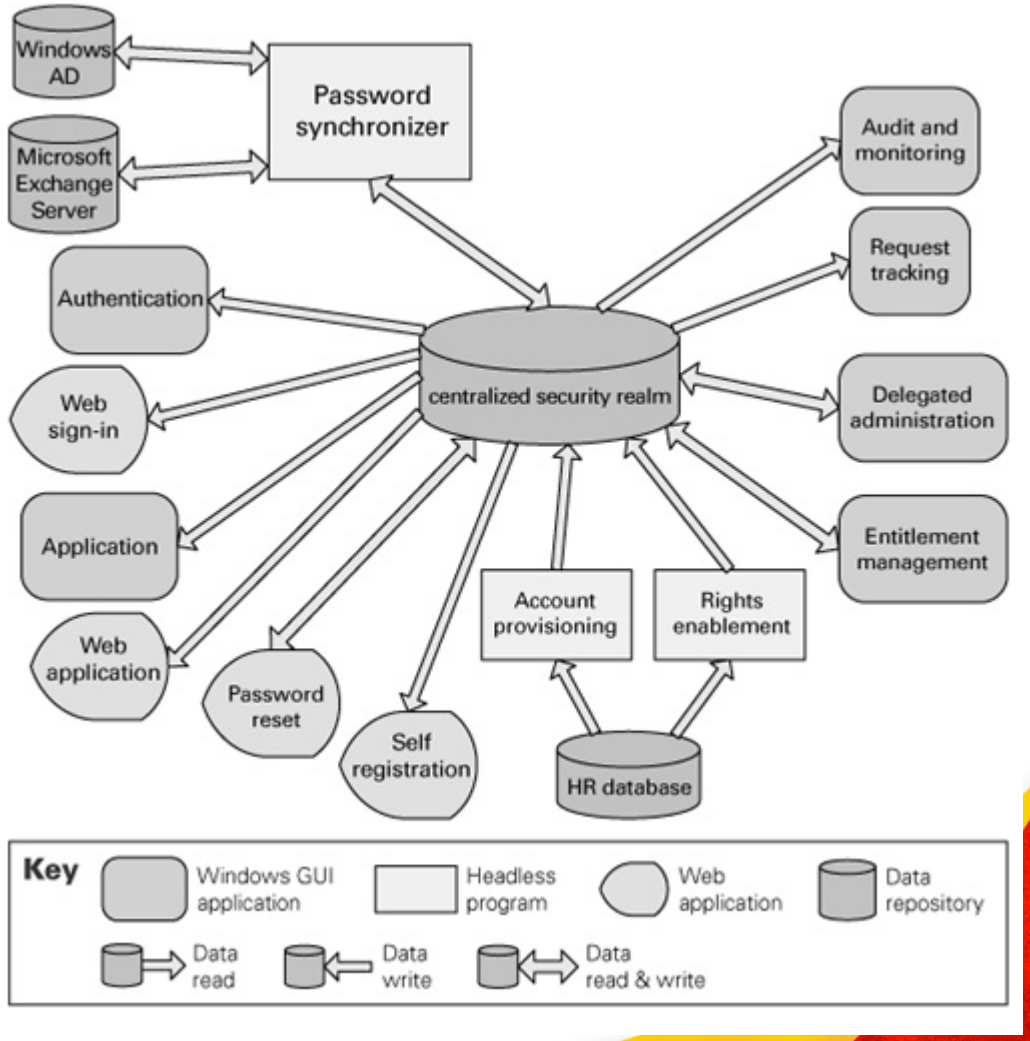
In the shared-data style, the pattern of interaction is dominated by the exchange of persistent data. The data has multiple accessors and at least one shared-data store for retaining persistent data.Database management systems and knowledge-based systems are examples of this style.

Elements	• Repository component. Properties include types of data stored, data performance-oriented properties, data distribution, number of accessors permitted. • Data accessor component • Data reading and writing connector. An important property is whether the connector is transactional or not.
Relations	Attachment relation determines which data accessors are connected to which data repositories.
Computational Model	Communication between data accessors is mediated by a shared data store. Control may be initiated by data accessors or the data store. Data is made persistent by the data store.
Constraints	Data accessors interact with the data models.
What it's for	• Allowing multiple components to access persistent data • Providing enhanced modifiability by decoupling data producers from data consumers
Relation to other styles	• This style has aspects in common with the client-server style, especially the multi-tiered client-server. In information management applications that use this style, the repository is often a relational database (DBMS). A bridge module or DBMS driver, built into the client components, provides database operations. • The shared-data style is closely related to the data model style. While a shared-data view of the system depicts the data repositories and their accessors, the data model shows how data is structured inside the repositories, in terms of data entities and their relations.

4. Repository Styles

Example of the Shared-Data Style

The shared-data diagram of an enterprise access-management system. The centralized security realm is a repository for user accounts, passwords, groups of users, roles, permissions, and related information. User IDs and passwords are synchronized with external repositories shown on the top left. The accounts of the enterprise employees are created/deactivated and permissions are granted/revoked based on status changes in HR database.



Crosscutting Issues for C&C Styles

- There are a number of concerns that relate to many C&C styles in a similar way. It is helpful to treat these as crosscutting issues, since the requirements for documenting them are similar for all styles.
- In these and other cases, the crosscutting issues can be documented by augmenting the element types of a style with additional semantic detail to clarify how instances of those types address the crosscutting issues. By adding this additional detail, we effectively create a specialized variant of the original style, because the augmentation will typically introduce new constraints on the components and connectors, their properties, and system topologies.

Crosscutting Issues for C&C Styles

Issue 1. Concurrent Communication Processes

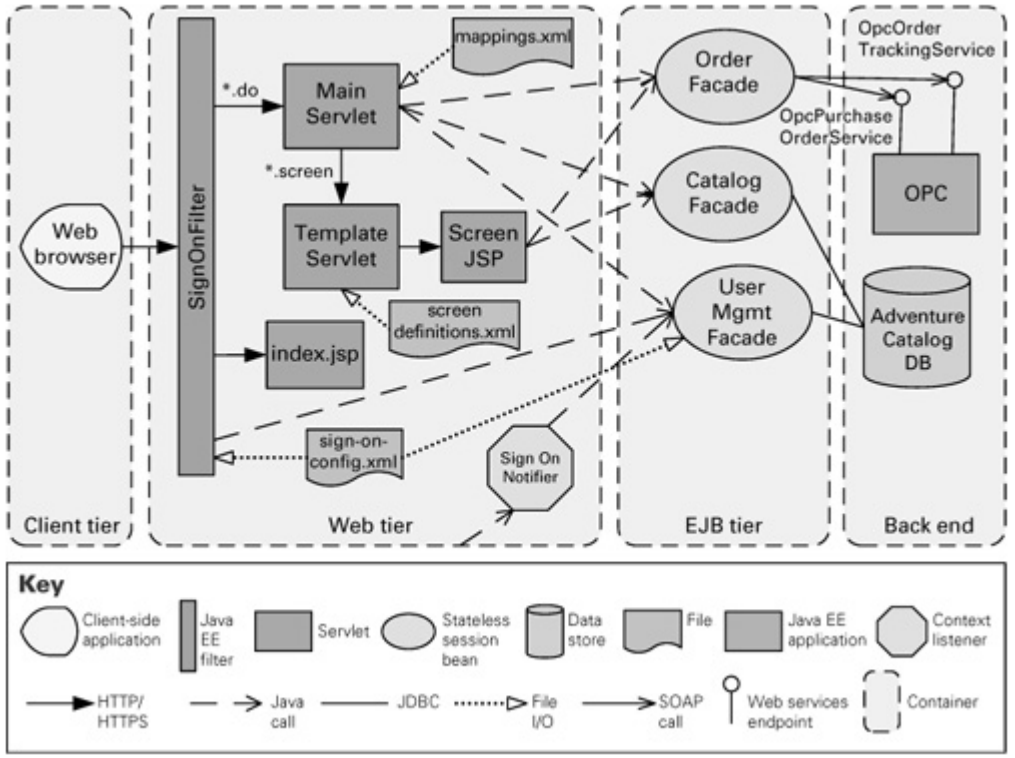
- Communicating processes are common in most large systems and necessary in all distributed systems. A communicating-processes variant of any C&C style can be obtained by stipulating that each component can execute as an independent process. For instance, clients and servers in a client-server style are usually independent processes. Similarly, a communicating-processes variant of the pipe-and-filter system would require that each filter run as a separate process. The connectors of a communicating-processes style need not change, although their implementation will need to support interprocess communication.
- Communicating processes are used to understand (1) which portions of the system could operate in parallel, (2) the bundling of components into processes, and (3) the threads of control within the system. Therefore this style variant can be used for analyzing performance and reliability, and for influencing how to deploy the software onto separate processors. Behavioral notations such as activity diagrams and sequence diagrams are particularly useful to understand interactions among elements running concurrently.

Crosscutting Issues for C&C Styles

Issue 2. Tiered Systems

- Another crosscutting issue is the use of tiers: aggregating components into hierarchical groupings and restricting communication paths between components in noncontiguous groups.
- The execution structures of many systems are organized as a set of logical groupings of components. Each grouping is termed a tier. The grouping of components into tiers may be based on a variety of criteria, such as the type of component, sharing the same execution environment, or having the same runtime purpose.
- Don’t confuse tiers with layers! Layering is a module style, while tiers apply to C&C styles. In other words, a layer is a grouping of implementation units while a tier is a grouping of runtime elements.

Diagram of the multi-tier view describing the Consumer Website Java EE application, which is part of the Adventure Builder system



Crosscutting Issues for C&C Styles

Issue 3. Dynamic Reconfiguration (Creation & Destruction)

- Many C&C styles allow components and connectors to be created or destroyed as the system is running. For example, new server instances might be created as the number of client requests increases in a client-server system. In a peer-to-peer system, new components may dynamically join the system by connecting to a peer in the peer-to-peer network. Because any style can in principle support the dynamic creation and destruction of elements, this is another crosscutting issue
- To document the dynamic aspects of an architecture, you should add several pieces of information, including the following:
  - What types of components or connectors within a style may be created or destroyed.
  - The mechanisms that are used to create, manage, or destroy elements. For example, component “factories” are a common mechanism for creating new components at runtime.
  - How many instances of a given component may exist at the same time. For example, some Web applications use a pool of instances of Web server components, and the number of instances in a pool is parameterized by a minimum and a maximum value.
  - What is the life cycle for different component types. Under what conditions new instances are created, activated, deactivated, and removed. For example, some styles require that all or part of a system be brought to a stable, “quiescent” state before new components can be added.

Acknowledgment

Lecture contents and pictures are taken from B4 Chapter 2 (Module Styles)