SEENG 401 Software Architecture

Lecture 6
Topic – Documenting Software Architecture
Sub-Topic – Component and Connector Views

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*

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

References: B4 – C3

Component and Connector (C&C) Views

Overview

Element	Description
Component	A C&C view shows elements that have some runtime presence like processes, objects, clients, servers, and data stores. These elements are called components. Components are principal processing units and data stores. • **Components have interfaces called ports.** A component has a set of ports which interacts with other components (via connectors). • **A port defines a specific point of potential interaction of a component with its environment.** A port usually has an explicit type, which defines the kind of behavior that can take place at that point of interaction. A component may have many ports of the same type. In this respect, ports differ from interfaces of modules, whose interfaces are never replicated. For example, a filter might have several input ports of the same type to handle multiple input streams, or a server might provide a number of request ports for client interactions. • **We can annotate a port with a number or range of numbers to indicate replication.** For example, a port annotated with "[3]" stands for three occurrences of that port. A port annotated with "[0..10]" means that there are from 0 to 10 instances of that port. That form is useful when defining component types, allowing component instances to bind the exact number, or for components that dynamically create new points of interaction.

Component and Connector (C&C) Views

Overview

Element	Description
Connector	A C&C view includes as elements the pathways of interaction like communication links and protocols, information flows, and access to shared storage. Such interactions are presented as connectors in C&C views. Connectors are pathways of interaction between components. Connectors have a set of roles that indicate how components may use a connector in interactions. • **<u>Simple examples of connectors are service invocation</u>**, asynchronous message queues, event multicast, and pipes that represent asynchronous, order-preserving data streams. But as we noted earlier, connectors often represent much more complex forms of interaction, such as a transaction-oriented communication channel between a database server and a client, or an enterprise service bus that mediates interactions between collections of service users and providers. • **<u>Connectors have roles</u>**, which are its interfaces, defining the ways in which the connector may be used by components to carry out interaction. For example, a client-server connector might have invokes-services and provides-services roles. A pipe might have writer and reader roles. Like component ports, connector roles differ from module interfaces in that they can be replicated, indicating how many components can be involved in its interaction. A publish-subscribe connector might have many instances of the publisher and subscriber roles. • **<u>A role typically defines the expectations of a participant in the interaction</u>**. For example, an invokes-services role might require that the service invoker initialize the connection before issuing any service requests. The semantics of the interaction represented by a connector is often documented as a protocol specification prescribing what patterns of events or actions are allowed to take place over the connector.

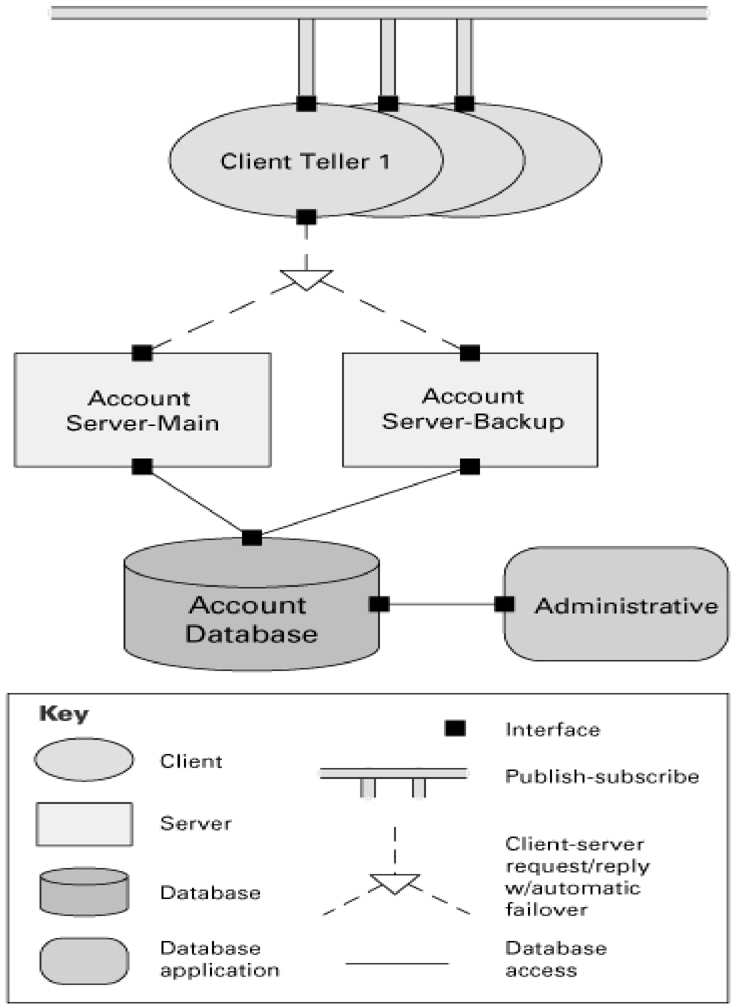
Component and Connector (C&C) Views

Overview

Key Points	Description
Relations	• Attachments: component ports are associated with connector roles to yield a graph of component and connectors • Interface delegation: in some situations, component ports are associated with one or more ports in an "internal" sub-architecture. Similarly, for the role a connector.
Constraints	• Components can be attached to only to connectors, not other components. • Connectors can be attached only to components, not other connectors. • Attachments can be made only between compatible ports and roles. • Interface delegation can be defined only between two compatible ports (or two compatible roles) • Connectors cannot appear in isolation; a connector must be attached to a component.
What C&C Views are for	• Show how the system works at runtime • Guiding development by specifying the structure and behavior or runtime elements • Helping architects and others to reason about runtime system quality attributes, such as performance, reliability, and availability

Component and Connector (C&C) Views

Example



A bird's-eye-view of a system as it appears at runtime. This system contains a shared repository that is accessed by servers and an administrative component. A set of client tellers can interact with the account servers and communicate among themselves through a publish-subscribe connector.

- Client-server connectors allow a set of concurrent clients to retrieve data synchronously via service requests. This variant of the client-server style supports transparent failover to a backup server.
- The database access connector supports transactional, authenticated access for reading, writing, and monitoring the database.
- The publish-subscribe connector supports asynchronous event announcement and notification.

Like components, complex connectors may in turn be decomposed into collections of components and connectors that describe the architectural substructure of those connectors. For example, a decomposition of the failover client-server connector of Figure left would probably include components that are responsible for buffering client requests, determining when a server has failed, and rerouting requests.

Component and Connector (C&C) Views

Advice for Connectors

- **Connectors need not be binary.** That is, they need not have exactly two roles. For example, a publish-subscribe connector might have an arbitrary number of publisher and subscriber roles. Even if the connector is ultimately implemented using binary connectors, such as a procedure call, it can be useful to adopt n-ary connector representations in a C&C view.
- **If a component's primary purpose is to mediate interaction between a set of components, consider representing it as a connector.** Such components are often best modeled as part of the communication infrastructure.
- **Connectors can—and often should—represent complex forms of interaction.** What looks like a semantically simple procedure call can be complex when carried out in a distributed setting, involving runtime protocols for time-outs, error handling, data marshaling, and locating the service provider—for example, as provided by SOAP.
- **Connectors embody a protocol of interaction.** When two or more components interact, they must obey conventions about order of interactions, locus of control, and handling of error conditions and time-outs. The protocol of interaction should be documented.

Component and Connector (C&C) Views

C&C Types and Advice

- When several components or connectors of a view share the same form and behavior (beyond what is specified in the style), define a common, application-specific type for them.
- Define application-specific types that cover all your components. This gives the reader one place to look for all component-and-connector details, rather than sometimes looking at type definitions and sometimes looking at instance information.
- The definition of a component type or connector type should explain the general computational nature and form of each of its instances.
- Application-specific types should provide enough information so that an architecture built from their instances can be correctly implemented and usefully analyzed.
- The component-and-connector types instantiated in a particular C&C view should be explained by referring to the appropriate style guide that enumerates and defines them, or through a catalog of application-specific types defined as part of the architecture.
- The definition of a component or connector type should characterize the number and type of interfaces (ports for components, roles for connectors) that instances of the type can have.
- A C&C view's primary presentation depicts only component-and-connector instances; no component types should appear in the view's primary presentation.
- When mapping between views, map modules to C&C types (not instances).

Relations in Component and Connector (C&C) Views

Primary Relation - Attachment

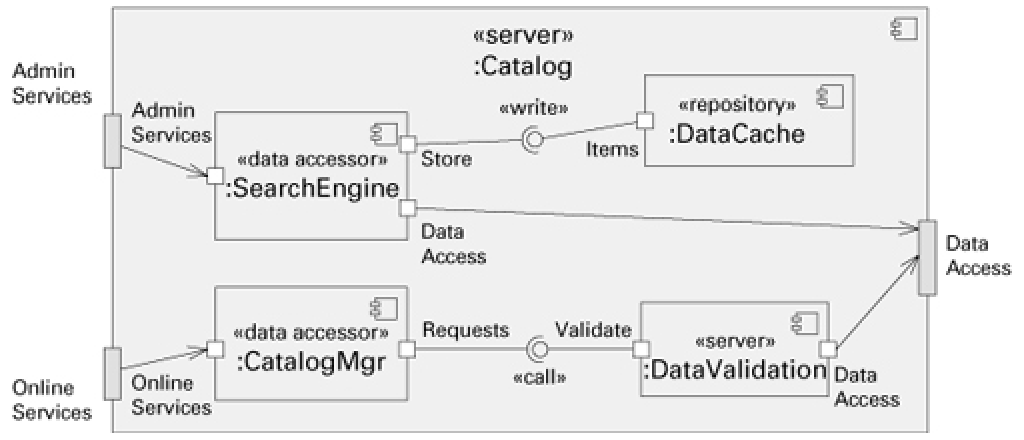
Overview	• Attachments indicate which connectors are attached to which components, thereby defining a system as a graph of components and connectors. Specifically, an attachment is denoted by associating (attaching) a component’s port to a connector’s role. • A valid attachment is one in which the ports and roles are compatible with each other, under the semantic constraints defined by the style. For example, in a call-return architecture, you should confirm that all “calls” ports are attached to some call-return connector. At a deeper semantic level, you should ensure that a port’s protocol is consistent with the behavior expected by the role to which it is attached.
Advice	Use the following guidelines when attaching components to connectors: 1. You can depict attachment simply by connecting the ports of components in the diagram. In this case, or in any case where the context makes clear what roles are being attached, you don’t need to represent roles explicitly in the diagram. 2. Attach a connector to a port of a component, not directly to a component. 3. If it might not be clear that it is valid to attach a given port to a given role, provide a justification in an annotation in the diagram or in the rationale section for the view. 4. Attaching connectors between ports annotated with a multiplicity factor (such as [5] or [0..10]) is a great source of ambiguity. For example, if you connect a port of multiplicity 3 to a port of multiplicity 22, what does that mean? If you connect two ports with the same multiplicity (greater than 1), which ports on one component are connected to which ports on the other? If you use this notation, explain what you mean.

Relations in Component and Connector (C&C) Views

Interface Delegation

When a component or connector has a subarchitecture, it is important to document the relationship between the internal structure and the external interfaces of that component or connector. The relationship can be documented using interface delegation relations. Such relations map internal ports to external ports (for components) or internal roles to external roles (for connectors). Some notations provide specific graphical elements to characterize this relationship.

A UML component diagram showing the subarchitecture of a component called Catalog. UML delegation connectors associate the ports of Catalog with the ports of internal components.



Properties in C&C Views

An element (component or connector) of a C&C view will have various associated properties. Every element should have a name and type. Additional properties depend on the type of component or connector. The properties are needed to guide the implementation and configuration of components and connectors, but the architect should also define values for the properties that support the intended analyses for the particular C&C view. For example, if the view will be used for performance analysis, latencies, queue capacities, and thread priorities may be necessary. Ports and roles also may have properties associated with them. For example, maximum sustainable request rates may be specified for a server port. Typical properties in a C&C view.

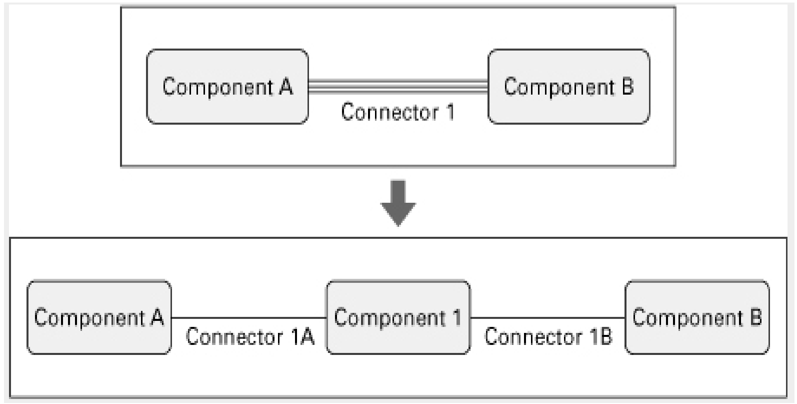
Reliability	What is the likelihood of failure for a given component or connector? This property might be used to help determine overall system reliability.
Performance	What kinds of response time will the component provide under what loads? What kinds of latencies and throughputs can be expected for a given connector? This property can be used with others to determine system properties such as response times, throughput, and buffering needs.
Resource requirements	What are the processing and storage needs of a component or a connector? This property can be used to determine whether a proposed hardware configuration will be adequate.
Functionality	What functions does an element perform? This property can be used to reason about overall computation performed by a system.
Security	Does a component or a connector enforce or provide security features, such as encryption, audit trails, or authentication? This property can be used to determine system security vulnerabilities.
Concurrency	Does this component execute as a separate process or thread? This property can help to analyze or simulate the performance of concurrent components and identify possible deadlocks.
Tier	For a tiered topology, what tier does the component reside in? This property helps to define the build and deployment procedures, as well as platform requirements for each tier.

Complex Connector vs Component

Needs for complex connectors:

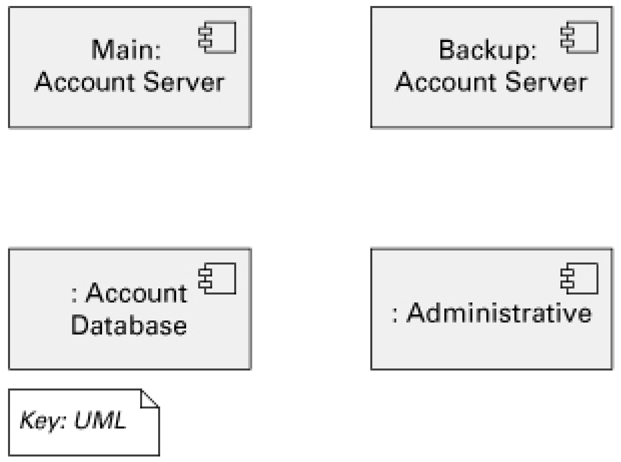
1. complex connectors are rarely realizable as a single mediating component. Although most connector mechanisms do involve runtime infrastructure that carries out the communication, that is not the only thing involved. In addition, a connector implementation requires initialization and finalization code; special treatment in the components that use the connector, such as using certain kinds of libraries; global operating system settings, such as registry entries; and others.
2. use of complex connector abstractions often supports analysis. For example, reasoning about a data flow system is greatly enhanced if the connectors are pipes rather than procedure calls or another mechanism, because well-understood calculi are available for analyzing the behavior of data flow graphs. Additionally, allowing complex connectors provides a single home where one can talk about their semantics.
3. using complex connectors helps convey an architect's design intent.
4. complex connector abstractions can significantly reduce clutter in an architecture model. Few would argue that the lower of the two diagrams in Figure is easier to understand. Magnify this many times in a more complex diagram, and it becomes obvious that clarity is served by using connectors to encapsulate details of interaction.

A complex connector and the alternative of representing it as a component with two simpler connectors

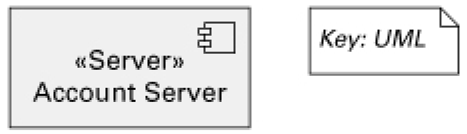


C&C Views
UML Notations

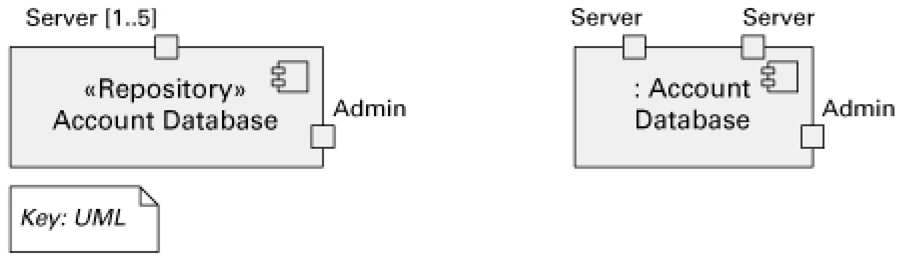
A UML representation of a portion of the C&C view originally presented in Figure. This fragment only shows how four components are represented in UML. Main and Backup are instances of the same component type (Account Server).



A UML representation of a C&C component type. The Account Server component type is a specialization of the Server component type from the client-server style.

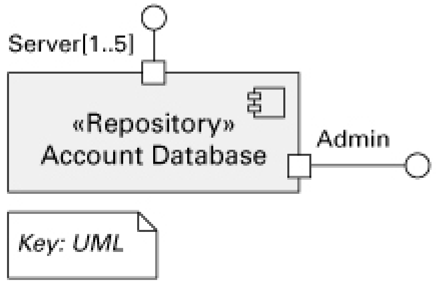


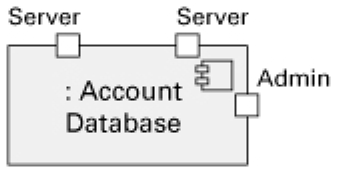
A UML representation of the ports on a C&C component type (left) and a component instance (right). The Account Database component type has two types of ports, Server and Admin (denoted by the boxes on the component's border). The Server port is defined with a multiplicity, meaning that multiple instances of the port are permitted on any corresponding component instance.



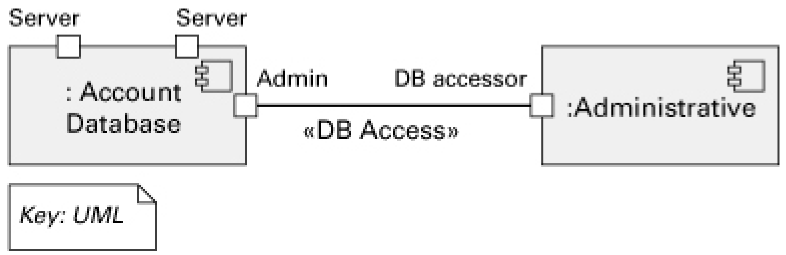
C&C Views
UML Notations

Each port on the Account Database type now includes one supplied interface (the lollipop), which can be further elaborated in UML by supplying additional information, such as methods or attributes. The instance of Account Database on the right has exactly two Server ports, and the interfaces are omitted.





A UML representation of a “simple” C&C connector between two components. The type of the connector is noted by a stereotype (<<DB Access>> in this case).



C&C Views

UML Notations

Figure 1. A UML representation of a “rich” C&C connector used to connect three components. The Publish-Subscribe connector is represented using a UML component. Its roles are represented using UML ports. Attachments between C&C ports and roles is represented using UML connectors between the respective UML ports.

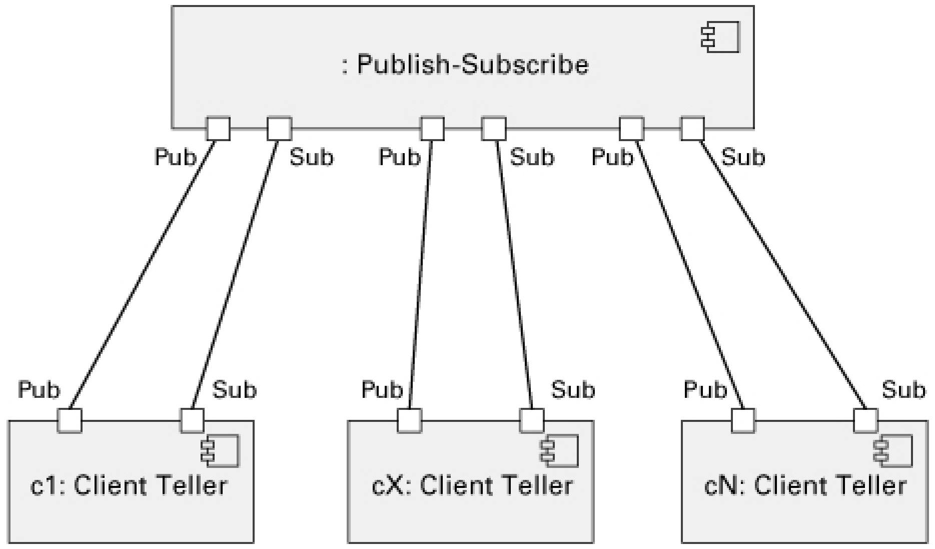
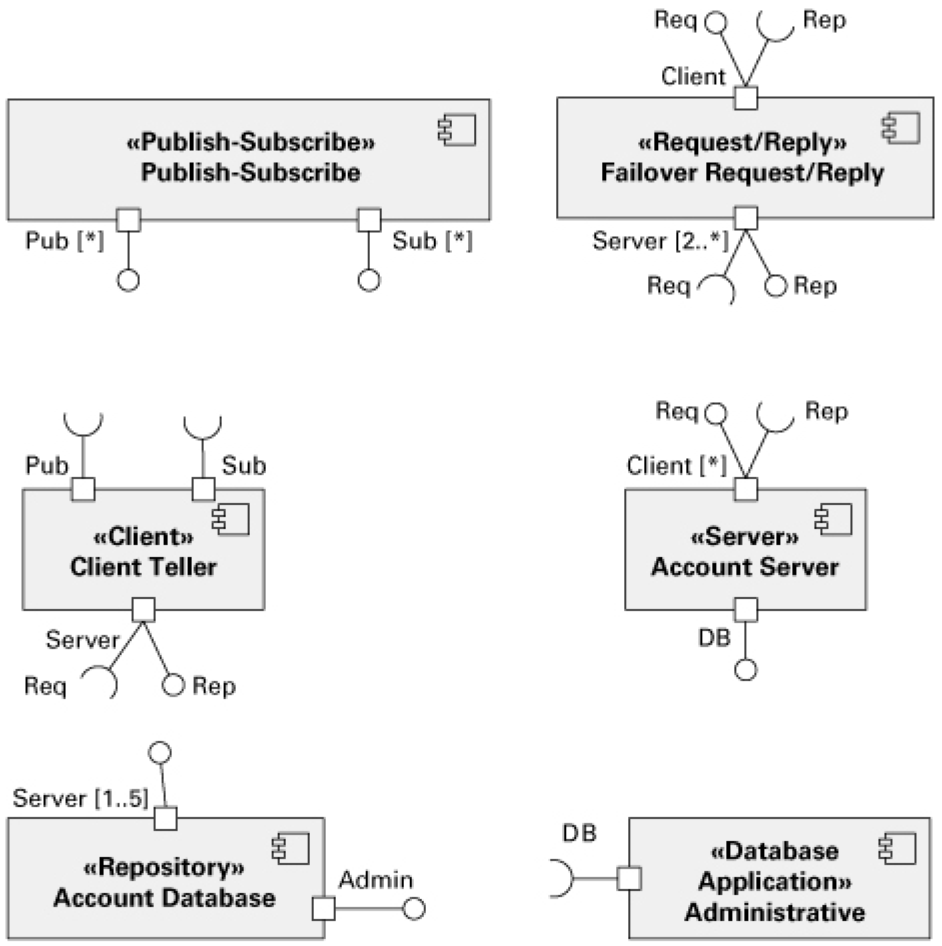


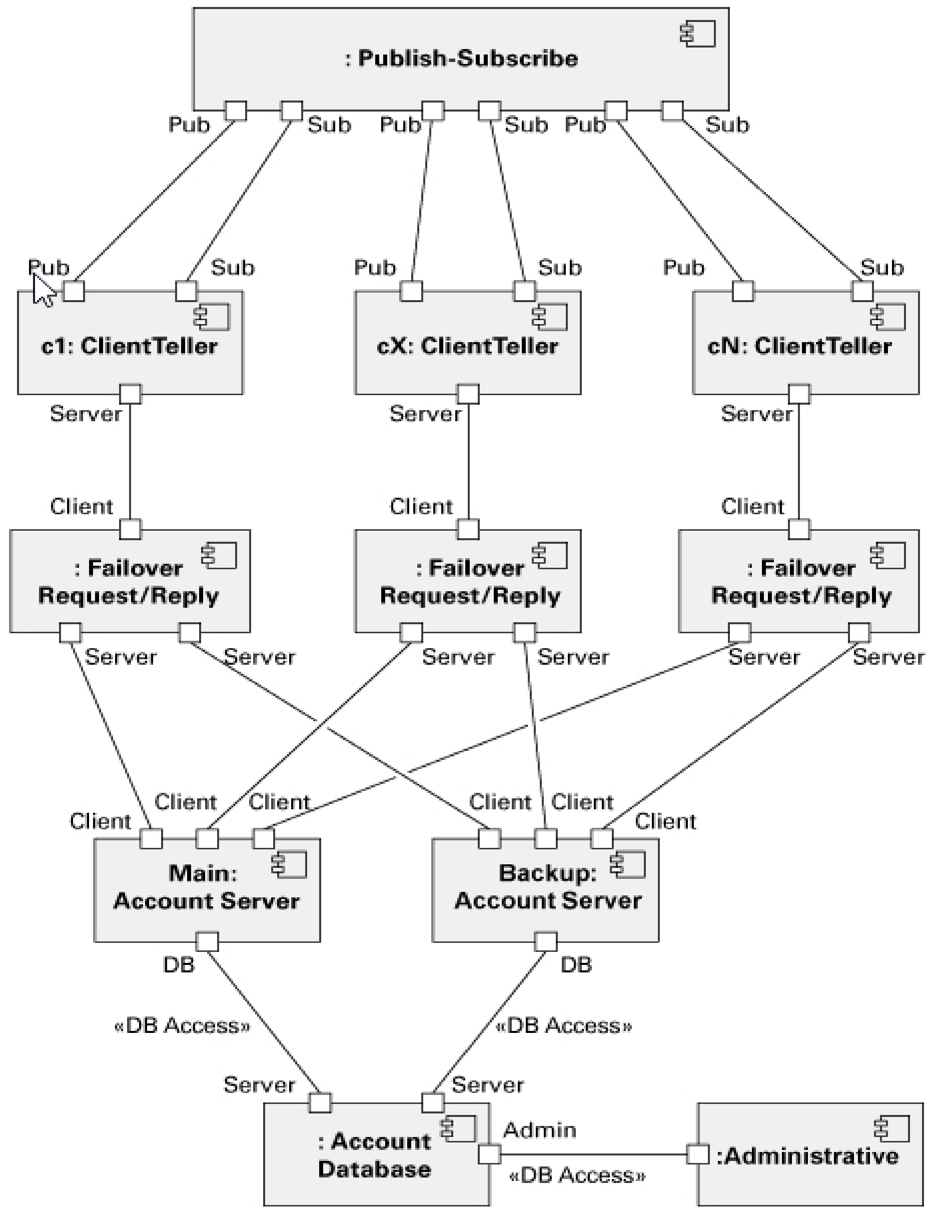
Figure 2. A UML representation of component-and-connector types for Figure 1. Each type uses a stereotype to link the view-specific subtypes to the types defined in the style guides.



C&C Views
UML Notations

Figure 2 defines the component-and-connector subtypes that are view specific. Each type uses a UML stereotype to identify the corresponding component or connector type defined in one of the three cited style guides. Multiplicities are attached to some of the ports to note where multiple connections are permitted and to set bounds on the number of connections. This information should be in the view's element catalog. Figure 3 shows the view's primary presentation, as represented using UML. Like the Publish-Subscribe connector, the Failover Request/Reply connector is represented using a UML component; this allows the details of the failover semantics to be formally documented, and it simplifies the representation of an n-ary connector

Figure 3. A UML representation of the primary presentation found in Figure 1



C&C Views
Relation to the other kinds of views

- Component-and-connector views differ from module views in fundamental ways. In particular, the elements of a C&C view represent instances of runtime entities, whereas the elements of a module view represent implementation entities. For example, consider a system that has 10 identical clients connected to a single server. That’s 11 components and 10 connectors—but exactly 2 modules (assuming the simplest mapping between views). An important consideration is how to relate the C&C and module views of a system. Often, the relationship between a system’s C&C views and its module views may be complex.
- The same code module might be executed by many of the elements of a C&C view.
- A single component of a C&C view might execute code defined by many modules.
- A C&C component might have many points of interaction with its environment, each defined by the same module interface.
- Since not every module is necessarily shown in every module view, a component in a C&C view may not map to any module in a particular module view at all.

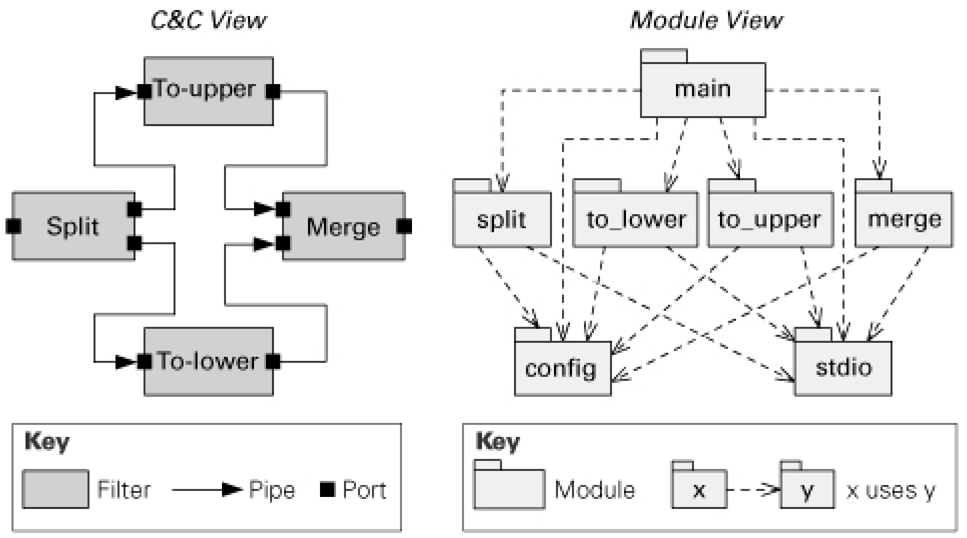
C&C Views
Relation to the other kinds of views

In many situations, however, module and C&C views have a more straightforward relationship. Indeed, systems that have natural correspondences between these two kinds of views are often much easier to understand, maintain, and extend. Here are two examples:

- Each component has a type that can be associated with an implementation module, such as a class. In this case the name of the component type will typically be taken to be the same as the corresponding module, making it trivial to relate the two views.
- Each module has a single runtime component associated with it, and the connectors are restricted to calls procedure connectors. This would be the case for an object-oriented implementation in which each class has a single instance.

C&C Views
Relation to the other kinds of views

Component-and-connector and module views of a simple system that accepts a stream of characters as input and produces a new stream of characters identical to the original but with uppercase and lowercase characters alternating



C&C View	Module View
System as a whole	Main
Split	Split, config, stdio
To-lower	to_lower, config, stdio
To-upper	to_upper, config, stdio
Merge	merge, config, stdio
Each pipe	stdio

Acknowledgment

Lecture contents and pictures are taken from B4