Lecture 5: Architectural Views
C&C Styles

Outline for Today

- Architecture views – C&C Views
Components and Connectors (C&C) Styles

- Elements
- Relations
- Properties

Components and Connectors (C&C) Styles - Elements

- **Components:**
  - principal units of runtime interaction, or
  - data stores
  - Components have *ports* to the outside world

- **Connectors:**
  - interaction mechanisms that connects components
  - allows components to interact with each others
  - connectors have *roles* that they can be called upon to fulfill

*ports and roles are just special names for the interfaces of components and connectors respectively*
Connectors can be very sophisticated and computationally intensive

- Protocol translation, format translation …
- Not always a simple procedure calls.
- Can be much more than that.

There’s only a single relation in C&C styles

- Attachment of components’ ports to connectors’ roles (interfaces with protocols)
- Lays out the computational topology of a system.
- Ports on a components interface are attached to connectors' roles.
- This is how components and connectors are tied together
Components and Connectors (C&C) Styles - Properties

- Name
- quality attributes (to help the analysis)
- others, depending on the style

Uses of the C&C - 1

- **Construction:**
  - for specifying the behavior that elements must exhibit when they work together

- **Education:**
  - as a starting point for the architect to show how the system works

- **Analysis:**
  - for reasoning about runtime system quality attributes such as?
    - performance, security, availability and reliability
Uses of the C&C - 2

- In terms of education and analysis, C&C styles can be used to answer questions such as:
  - What are the major executing components and how do they interact?
  - What are the major shared-data stores?
  - Which parts of the system are replicated?
  - How does data progress through the system?
  - Which parts of the system can run in parallel?
  - How can the system’s state change as it executes?
  - Can components come into existence and be destroyed?
  - Can connections exist, be created, and destroyed as the system executes?

C&C Styles

- There are many
- Grouped into four major types:
  - Dataflow
  - Call-Return
  - Event-Based
  - Repository
# C&C Styles

- **Dataflow**
  - Functional transformations process inputs to produce outputs
  - *Example: pipe-and-filter, patch sequential*
- **Call-Return**
  - Components receive control-and-data from other components, and then when they're finished executing, return control to the component that invoked them
  - *Example: Client-server, peer-to-peer, SOA*
- **Event-Based**
  - Control transfers not through explicit calls or explicit invocations but through the occurrence of events.
  - Components can publish and/or subscribe to events
  - *Example: Publish-subscribe, SOA*
- **Repository**
  - Large stores of persistent data.
  - large central, or several central databases in which persistent data is stored
  - *Example: Shared-data, blackboard*

## Pipe-and-Filter Style – 1

- **Data flow style**
- **Overview of Pipe-Filter Style**
  - Data is transformed serially from a system’s external inputs to its external outputs through a series of transformations performed by its filters
- **Elements:**
  - component type: *filter*, which transforms data on its input ports to data written on its output ports.
  - connector type: *pipe*, a unidirectional data conduit that conveys data from a filter’s output ports to another filter’s input ports. It *preserves the order and value of data.*
- **Relations:**
  - attachment of pipes to filters
    - associates filter output ports with data-in roles of a pipe, and filter input ports with data-out roles of pipes
Pipe-and-Filter Style – 1

- **Properties**
  - How end-of-data is signaled
  - The capacity of the pipes (throughput and performance parameters)
  - Whether or not each filter is a separate process

Pipe-and-Filter Style - 2

- **Constraints:**
  - Pipes connect filters.
  - Connected filters must agree on the type of data being passed along the connecting pipe.
  - Branches? Loops?

- **What it’s for:**
  - Improving reuse (each filter is independent)
  - Supporting functional composition data analysis
  - Improving throughput in systems in which data is transformed serially (vs. patch sequential)

\[ y = C(B(A(x))) \]
Pipe-and-Filter in UML

Batch-Sequential vs. Pipe-Filter
Client-Server Overview

- Client-Server is a type of **Call-Return Style** that represents a computational model in which components provide a set of “services” that may be invoked by other components.

Client-Server Elements

- Elements:
  - **client**, which is a component that invokes services of a server component
  - **server**, which is a component that provides services to client components
  - **request/reply connector**, which is used by a client to invoke services on a server.
    - Request/reply connectors have two roles:
      - a request role and a reply role.
Relations and Properties

- **Relations:**
  - attachment of clients to servers.

- **Properties:** Element properties may include
  - information about the nature of the server ports (such as how many clients can connect),
  - performance characteristics (such as maximum rates of service invocation), and
  - whether clients and server can be introduced dynamically.

- Connector properties may include whether the calls are local or remote and whether data is encrypted.

Constraints

- Clients must be connected to servers through request/reply connectors.
- Server components may be clients to other servers.
- Specializations may impose restrictions:
  - Number of attachments to a given port
  - Components may be arranged in tiers.
What Client-Server Style is for

- Promoting modifiability and reuse by factoring out common services
- Improving scalability and availability in case server replication is in place
- Analyzing dependability, security, and throughput

ATM Example
Service-Oriented Architecture Style – 1

- This is both a Call-Return and Event-Based Style
  - This depends on the type of connector

- **Overview:**
  - Service users interact with service providers in a network through published interfaces that allow interoperability.

- **Elements:**
  - Component types:
    - service consumers, service providers
    - infrastructure components (Enterprise service bus; a specialized component that's a registry of services, or maybe an orchestration server which orchestrates the ways in which servers carry out their duties)
  - Connector types:
    - call-return (such as with SOAP, REST)
    - asynchronous messaging (event-based)

*A service is a self-contained, distributed component with a published interface that stresses interoperability, is discoverable, and can be dynamically bound.*

---

Service-Oriented Architecture Style – 2

- **Relations:**
  - Attachment of a service call to a service endpoint

- **Properties:**
  - Include performance, reliability, availability, **cost**, and others that can be associated to services

- **Constraints:**
  - Service consumers are connected to service providers.
  - A service consumer may also be another service provider.
  - Infrastructure components may mediate the interaction between service consumers and service providers:
    - registry of services: naming and location of services
    - ESB: routing, data and format transformation, technology adapters
    - orchestration server: coordinates service invocations based on a scripted workflow
What is SOA Good for

- Distributed applications
- Interoperability of components developed in different languages and platforms
- Integration of external components and legacy systems

Uses of the Service Oriented Styles

Example

Source: Adaptation of Java Adventure Builder Reference application (Oracle 2010).

Event-based or call-return?
Publish-Subscribe Style – 1

- An Event-Based style
- **Overview:** Components interact via events that they announce and to which they subscribe
- **Elements:**
  - Any C&C component with at least one publish or subscribe port
  - Publish-subscribe event distributor (e.g., event bus), which will have “announce” and “listen” roles for components that wish to publish and/or subscribe to events

![Diagram of publish-subscribe event distributor](image)

Publish-Subscribe Style – 2

- **Relations:** Attachment relation
  - associates components with the publish-subscribe event distributor
  - prescribes which components announce events and which components have registered to receive events.
- **Properties:** include
  - events that are announced and/or subscribed to
  - conditions under which an announcer is blocked.
Publish-Subscribe Style – 3

- **Constraints:**
  - An event distributor is connected to all other components.
  - Publish ports are attached to announce roles.
  - Subscribe ports are attached to listen roles.
  - A component may be both a publisher and a subscriber, by having ports of both types.

Publish-Subscribe Style – 4

- **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

- **Variations:**
  - may restrict which components can listen to which events,
  - whether a component can listen to its own events, and
  - whether there can be only one or multiple instances of the pub-sub connector
  - The event distributor may be depicted as a bus (that is, a connector) or a component.
Shared-Data Style – 1

- **Repository style**
  - **Overview:**
    - Data accessor components read and write to a shared data store.
  - **Elements:**
    - component types: data stores and accessors
    - connector types: data reading and writing
  - **Relations:**
    - attachment

Shared-Data Style - 2

- **Constraints:**
  - The data store is attached to the data accessors via connectors.
- **What it’s for:**
  - sharing among multiple accessors of persistent data items
Uses of Shared Data Style

- Examples

Crosscutting Issues for C&C Styles:
- Communicating Processes
- Tiers
- Dynamic Creation and Destruction
Communicating Processes - 1

- 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.
  - Clients and servers in a Client-server style usually run as independent processes.
  - 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 inter-process communication.

Communicating Processes - 2

- Communicating processes are used to understand:
  - which portions of the system could operate in parallel
  - the bundling of components into processes
  - the threads of control within the system

- Additional documentation for communicating-processes includes
  - mechanisms for starting, stopping, and synchronizing a set of processes or threads
  - priority of the processes, which influences scheduling
  - preemptability, whether the execution of a concurrent unit may be preempted by another concurrent unit
  - timing parameters, such as period and deadline
  - additional components, such as watchdog timers and schedulers, for monitoring and controlling concurrency
  - use of shared resources, lock mechanisms, and deadlock prevention or detection techniques.
Execution structures of many systems are organized as a set of logical groupings of components. Each grouping is a tier.
- The use of tiers may be applied to any C&C style.
- In practice, tiers are most often used in the context of Client-server styles.

Grouping of components into tiers may be based on
- the type of component
- sharing the same execution environment
- having the same runtime purpose

Tiers induce topological constraints that restrict which components may communicate with other components.
- In the typical topology
  - the Client tier interacts with the Web tier.
  - the Web tier interacts with the Business Logic tier.
  - the Business Logic tier interacts with the Backend/Database tier.

Interactions are unidirectional, but one tier may bypass another.
Client tier components can be “thin” or “fat.”
Many C&C styles allow components and connectors to be created or destroyed as the system is running.

**Example:**
- New server instances might be created as the number of client requests increases in a client-server system.

To document dynamic aspects, include:
- what types of components or connectors within a style may be dynamically created or destroyed
- what mechanisms are used to create, manage, or destroy elements (e.g., component factories, pools of component instances)
- how many instances of a component may exist at a time
- the life cycle for different component types:
  - under what conditions new instances are created, activated, deactivated, and removed
The primary function of a context diagram is to show what’s in and what’s out of the system (or part of the system) that we’re considering at the moment.
What’s in a Context Diagram?

- A context diagram depicts the
  1. system (or part of the system) being described
  2. environment in which system functions
  3. relations or interactions between the system and its environment

Where Do We Document Context Diagrams?

[Diagram showing a template for a view with sections on primary presentation, element catalog, context diagram, variability guide, and rationale.]
Context Diagrams in the “Language” of the View - 1

- A context diagram should show the context using the language (i.e., element and relation types) of the view in which it appears.

- In a C&C view, a context diagram shows the pathways of runtime interaction between the system and external elements.

This is context diagram for a C&C view. What would a context diagram for other kinds of views look like?

Context Diagrams in the “Language” of the View - 2

- In the language of a Layered view, a context diagram shows the system as a layer with any other existing layers above or below it.

- In the language of a Decomposition view, a context diagram shows the system being developed as a module and how it is nested along with other modules inside of a larger system.
Context Diagrams in the “Language” of the View - 3

<table>
<thead>
<tr>
<th>In the “language” of…</th>
<th>…a context diagram shows…</th>
</tr>
</thead>
<tbody>
<tr>
<td>…a Uses view</td>
<td>any external modules that use or are used by our system</td>
</tr>
<tr>
<td>…a Deployment view</td>
<td>any other systems that share the same environmental resources as our system</td>
</tr>
<tr>
<td>…a Data model view</td>
<td>relations to data entities outside the system</td>
</tr>
</tbody>
</table>

If a context diagram doesn’t apply for a view, omit it by indicating “not applicable.”

Notations for Context Diagrams - 1

- **Informal:**
  - box-and-line drawings that should clearly define the system (or the part of it being defined)
  - tables that list external entities and any interactions with them (such as messages passed). In this case, the context diagram isn’t really a diagram!

- **Semiformal:**
  - Structured Analysis notation (Demarco, Gane-Sarson, Yourdon), circa 1979
  - UML:
    - no explicit context diagram
    - several techniques possible
Notations for Context Diagrams - 2

- Use normal UML constructs to show the system and its relation to external elements. Use a <<stereotype>> to designate "our system." Examples are:
  - component diagram for C&C context
  - nested packages for module decomposition context
  - <<layer>> packages and <<allowed to use>> for layered context
  - class diagram for generalization context

- For C&C context diagrams, you can also use use case diagrams as follows:
  - actors represent the environment
  - association of actors with use cases stands for communication
  - annotations on associations can explain information flows
  - a system bounding box is optional (but recommended)