Outline for Today

- Team Presentations: Tactics
- Architecture views
  - Module view
Architectural Views - 1

Many projects make the mistake of trying to impose a single partition in multiple component domains, such as equating threads with objects, which are equated with modules, which in turn are equated with files. Such an approach never succeeds…

-Jazayeri, Ran, and van der Linden

Architectural Views - 2

Objectives:
- define the terms view and style
- name three types of views
- describe how each type of view is used
- describe how views and styles are related
Architectural Views

- An architecture is a multidimensional construct, too complex to be seen all at once.

- Systems comprise many structures that show:
  - how components and connectors work at runtime
  - processes and how they synchronize
  - programs and how they call or send data to each other
  - composition/decomposition of modules
  - mapping of modules to implementation units
  - how software is deployed on hardware
  - how teams cooperate

- Views are representations of structures.
- They help manage complexity

View - Definition

- A view is a representation of a set of system elements and the relations associated with them.
- Only some system elements are shown.
- A view constrains the types of elements, relations, and properties that are represented in that view.
In box-and-line diagrams, another way of asking what the boxes and lines mean is “What element types and relation types are you showing?”

In other words “What view are you showing?”

Views solve our problem of not being able to show everything at once

---

**View-Based Documentation**

- Views give us our basic principle of architecture documentation:

  *Documenting a software architecture is a matter of documenting the relevant views and then adding information that applies to more than one view.*

- Break down the documentation into smaller, more manageable tasks
Right set of views?

- No such set exists
- You choose the best set of views for the particular situation or system
- *ISO/IEC 42010: prescribes creating your own views to serve specific stakeholder concerns
- Right set of views depends on:
  - the structures that are inherent in the software
  - who the stakeholders are and how they will use the documentation

*Derived from ANSI/IEEE 1471-2000 (Institute of Electrical and Electronics Engineers, 2000).

Recognizing the Types of Views - 1

- An architect must consider the system in *three* ways:
  1. How it is structured as a set of implementation units
  2. How it is structured as a set of elements having runtime behaviors and interactions
  3. How it relates to non-software structures in its environment
Recognizing the Types of Views - 2

Different types of views show different types of information:

- **Module views** show how the system is structured as a set of implementation units
- **Component-and-connector views** show how the system is structured as a set of elements with runtime behaviors and interactions
- **Allocation views** show how the system relates to non-software structures in its environment.

---

Modules vs. Components

The following client-server system consists of two modules (units of implementation) but 11 components (units of runtime interactions and behaviors)

- **Decomposition view**
- **Client-server view**
Architectural Styles - 1

- Recurring solution forms that have been widely observed in different systems
- These forms or patterns are worth capturing because they have known properties and can be reused

- *An architectural style is a specialization of element and relation types, together with a set of constraints on how they can be used*

Architectural Styles - 2

- Styles can be discussed independently of any system
  - Two or more systems can use the same style

- Choice of style is an important architectural decision
  - Different styles are more (or less) helpful in achieving quality attribute goals

- Most styles we read about belong to the Component-and-connector (C&C) styles: they show units of runtime interaction.
  - But module styles and allocation styles exist as well.

- Styles exist in every system.
  - Every system is built using some specialization of element and relation types together with a set of constraints on how they can be used.
  - The styles in published catalogs are ones that are reused across many systems.
Views and Styles

- A view is a style that is applied to a system by the architect.

Example #1:
- A **Client-server style** prescribes a system built with elements that are clients and servers, with constraints on their topology and interactions.
- A **client-server view** shows that system S is built with three servers and 34 clients, each with a name and specific properties. This view shows how they actually communicate and interact with each other.

Example #2:
- A **Layered style** prescribes a system built with elements that are layers, with constraints on their topology and interactions. The style does not define specific layers.
- A **layered view** shows that system S is built with seven named layers, and it gives the specific responsibility and interface of each one.

Heterogeneous Systems

- Systems can contain large numbers of dissimilar or diverse elements, even in the same category of view.
  - Because of that complexity, their architectures are guaranteed to use more than one style.

- Heterogeneity can occur in multiple ways:
  1. Different areas of the system exhibit different styles.
  2. An element of one style is composed of elements arranged in another style.
Different areas of the system might exhibit different styles.
- Example: A Pipe-and-filter style feeds a Shared data repository.

An element of one style might be composed of elements arranged in another style.
- Example: A service provider in the SOA style is structured internally using a Multi-tier style.
Template for Documenting a view

- Section 1: Primary presentation
  - (usually graphical, e.g., a UML diagram)

- Sections 2-5: Supporting documentation
  - Section 3 is the meat of a view documentation
  - We will discuss sections 4-5 later in the semester

Module View Styles

*a program’s runtime structure often bears little resemblance to its code structure... In fact, the two structures are largely independent.*

-Gamma, Helms, Johnson, and Vlissides

- Objectives:
  - identify three kinds of notations: informal, semiformal, and formal
  - describe the role of element properties
  - identify a number of module styles
  - describe how to document views in each of the styles
Module View Styles

- Decomposition style
- Uses style
- Generalization style
- Layered style
- Data model style

Collection of Styles

- **Remember:** Systems are composed of more than one style.
- Our focus here is on “pure” styles.
- These form the conceptual foundation for the “hybrid” or mixed styles that most systems exhibit.
How Styles Are Presented

1. overview of the style
2. description of the style’s element types, relation types, and properties
3. constraints (topological restrictions)
4. what the style is good for
5. useful notations and analysis techniques for the style
6. examples of views using this style

Elements Properties - 1

- To document a view, you must decide which properties of elements to document.

- Properties characterize the elements and their relations;

- Properties almost always include the name of the element as well as some description of its role or responsibility in the architecture.
Elements Properties - 2

- If you care about quality attribute $x$, define and document properties that will let you analyze for $x$ in the views that are related to achieving $x$.
- Examples:
  - For analyzing an architecture for performance, useful properties include best-and worst-case response times or the maximum number of events an element can service per time unit.
  - For analyzing an architecture for security, useful properties will highlight levels of encryption and authorization rules for different elements and relations.
  - For analyzing and architecture for modifiability/testability, useful properties will focus on size of module (LOC or function points) and/or complexity of the code elements.

Notations - 1

- **Informal notation**
  - Graphical depictions use general-purpose diagramming and editing tools and visual conventions.
  - Semantics are characterized in natural language (if at all).
  - Informal notations cannot be formally analyzed.
  - Example: boxes and lines drawn with PowerPoint.
**Notations - 2**

- **Semiformal notation**
  - This standardized notation prescribes graphical elements and rules of construction.
  - It does not provide a complete semantic treatment of the meaning of those elements.
  - Simple analysis can tell if a description satisfies syntactic properties.
  - Example: UML

**Notations - 3**

- **Formal notation**
  - Notation has a precise (usually mathematically based) semantics.
  - Formal analysis of both syntax and semantics is possible.
  - Automated tools provide useful analysis of the architecture and/or automation to assist in code generation.
  - Example: AADL
Module Styles

- **Elements**: modules.
  - A module is an implementation unit that implements a set of responsibilities

- **Relations**: Relations among modules include the following:
  - **A is part of B**: This defines a part-whole relation among modules
  - **A depends on B**: This defines a dependency relation among modules
  - **A is a B**: This defines specialization and generalization relations among modules

- **Properties**: Properties of modules include name, responsibilities, and the visibility of the module and its interface. Other properties are style specific

What Are Module Styles Used For?

- **Construction**:
  - Blueprints for the code.
  - Assigned to teams for implementation
  - Basis for subsequent design (e.g., interfaces), subsets, and deployment procedures.

- **Analysis**:
  - Traceability and impact analysis
  - Project management, budgeting, planning, and tracking is based on modules

- **Education**: A software developer can learn the project’s structure through understanding Module views.
Module View Styles

- Decomposition style
- Uses style
- Generalization style
- Layered style
- Data model style

Decomposition Style

- **Overview:** documents how system responsibilities are partitioned across modules and how those modules are decomposed into sub-modules
- **Elements:** modules
- **Relations:** “is part of.”
- **The criteria for decomposition:**
  - achievement of modifiability
  - build versus buy (or reuse of components)
  - software product lines: common versus unique parts
  - developers’ skills
- **Constraints:**
  - A child can have only one parent.
- **What it’s for:**
  - assigning responsibilities to modules as a prelude to downstream work
  - conducting change/impact analysis
  - developing work assignments
  - communicating to newcomers, in digestible chunks, how the software is organized
Decomposition Style: Notations

- **Informal:**
  - box and line, nested boxes
  - textual outline
  - Table

- **Semiformal:**
  - UML—nesting is the preferred notation
  - (Although you may encounter a UML diagram like this, the composition arrow shows part-whole relations in the *object* domain only.)

---

Decomposition Style: Textual Example

1. **Behavior-Hiding Module**
   - Function Driver Module
   - Air Data Computer Module
   - Audible Signal Module
   - Computer Part Signal Module
   - Doppler Radar Module
   - Flight Information Display Module
   - Forward Looking Radar Module
   - Head-Up Display Module
   - Inertial Measurement Set Module
   - Panel Module
   - Projected Map Display Set Module
   - Shipboard Inertial Nav. Sys. Mod.
   - Visual Indicator Module
   - Weapon Release Module
   - Ground Test Module
   - Shared Services Module
   - Mode Determination Module
   - Panel I/O Support Module
   - Shared Subroutine Module
   - Stage Director Module
   - System Value Module

2. **Software Decision Module**
   - Application Data Type Module
   - Numeric Data Type Module
   - State Transition Event Module
   - Data Banker Module
   - Singular Values Module
   - Complex Event Module
   - Filter Behavior Module
   - Physical Models Module
   - Aircraft Motion Module
   - Earth Characteristics Module
   - Human Factors Module
   - Target Behavior Module
   - Weapon Behavior Module
   - Software Utility Module
   - Power-Up Initialization Module
   - Numerical Algorithms Module
   - System Generation Module
   - System Generation Parameter Module
   - Support Software Module

---

*Source: Software Architecture in Practice, Software Cost Reduction (“A-7E”) Architecture (Bass 2003, Chapter 3)*
Overview: tells the developer what other modules must exist for this portion of the system to work correctly

Elements: modules

Relations: "uses," a specialization of "depends on"

- A uses B if A depends on the presence of a correctly functioning B to satisfy its (A's) own requirements.

Constraints:

- None.
- However, large loops are undesirable, as they can cause problems with incremental system delivery.

What it's for:

- planning incremental development
- use in system extensions and subsets
- debugging and testing
- gauging the effects of specific changes
Uses Style: UML Example

Notation for Uses Style: Dependency Structure Matrix (DSM)

This DSM corresponds to the UML diagram on the previous slide.

Key: ‘1’ means module in column uses module in row
What do we need if we want C in our next increment?

Now what do we need if we want C in our next increment?
Generalization Style

- **Overview**: The basis for object-oriented designs
  - this style captures the “is a” relations among elements of the system.
- **Elements**: modules (classes or objects)
- **Relations**: generalization, an “is a” relation
- **Properties**: abstract? The module is abstract if it does not contain a complete implementation.
- **Constraints**: Cycles are prohibited.
- **What it’s for**:
  - incrementally describing evolution and extension
  - capturing commonalities, with variations as children
  - supporting reuse

Generalization Style: Notations

- **Formal, semiformal**:
  - programming languages
  - UML
Layered Style - 1

- **Overview**: documents “allowed to use” relations among elements of the system
- **Elements**: layers, a virtual machine
- **Relations**: “allowed to use,” a specialization of the “depends on” relation
  - Recall that A uses B if A’s correctness depends on the presence of a correct B.
- **Constraints**:
  - Every piece of software is assigned to exactly one layer.
  - There are at least two layers (typically three or more).
  - Software in a layer is allowed to use software in {any lower layer, next lower layer}
    - Which one is correct?
  - Software in a layer {is, is not} allowed to use other software in the same layer.
    - Which one is correct?

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>C</th>
</tr>
</thead>
</table>
```

Layered Style - 2

- **What it’s for**:
  - promoting portability and modifiability
  - fielding subsets and incremental development
  - separation of concerns
  - reuse
- **Variations**:
  - segmented layers: dividing a layer into segments (or submodules), with “allowed to use” relations between those segments and segments from other layers
Layered Style: Informal Notations - 1

Are these architecturally equivalent?

Layered Style: Informal Notations - 2

interface visibility

boxes and arrows
Layered Style in UML: Packages

Layered Style: Example 1

Source: OSGi Architecture (OSGi Alliance 2007, p. 11).
Layered Style: Example 2

Overview: This style models important data entities and the relations among them.

Elements: data entity, which holds information that needs to be stored or somehow represented in the system.

Relations:
- one-to-one, one-to-many, many-to-one, and many-to-many relationships, which are associations between data entities
- generalization/specialization, which indicate an is-a relation between entities

Constraints: normalization may impose restrictions on intra and inter-entity dependencies
Data Model Style - 2

What it’s for:
- describing the structure of the data used in the system
- performing impact analysis of changes to the data model; extensibility analysis
- enforcing data quality by avoiding redundancy and inconsistency
- guiding implementation of modules that access the data
- guiding database performance optimizations
- prior to design, capturing the result of domain analysis

Data Model Style: Notations

Semiformal:
- UML class diagram
- entity-relationship diagram (ERD)
Summary

- A module is any code-related unit that the architect feels is useful to contemplate.
- The type of implementation unit depends on the type of implementation platform.
- To document a view, you must decide which properties of elements to document.
- Properties characterize the elements and their relations.
- There are three kinds of notations:
  - Informal notations use general-purpose diagramming and editing tools and visual conventions.
  - Semiformal notations prescribe graphical elements and rules of construction.
  - Formal notations have precise (usually mathematically based) semantics.

Next Week

- Quiz: Reading Chapters 1, 2, 3, & 4 of Documenting Software Architecture
- C&C styles
- AADL work
  - Bring your laptop
  - Will send instruction on how to install tool
- Team work
  - Documenting the module view of the architecture document for the semester project
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.

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
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.

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.
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?

---

<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)ss diagram for generalization context
Project Team Assignment

- Complete the sections related to the Module views.
- Due on Tuesday 02/28/2017
  - @ 5:59 pm