Architectural Design—Outline

Prev lecture—general design principles. Today—architectural design

1. What is a software architecture

2. Components, Connectors, and Configurations


EXAMPLE: A 3-tier Architecture for an Internet auction system

Requirements: A web-based system for conducting internet-based auctions. Live, interactive bidding, with a human auctioneer controlling the proceedings.
Models of architectural design!
Why Create Architectural Models?

Consider that architects of buildings create models: blue prints, wiring diagrams, plumbing diagrams, structural diagrams etc. why?

1. Models help describe how the design artifact meets requirements.
2. Models help spot inconsistencies, missing details, etc.
3. Models can be used to derive and predict properties of the artifact.
4. Models can be used as a reference during construction.
5. Models can be used to automatically generate the final system!

So, what does an architectural model have...?
Architectures, and Architectural design

Adapted From Garlan & Shaw:

• Software architecture:
  – Addresses system-level issues: scale, capacity, throughput, consistency, inter-operability, security, distribution etc.
  – Defines a system in terms of components and interactions between components (the highest level of such a definition)
  – Shows correspondence between requirements and the elements of the constructed system

• Architectural Design:
  – Selects components which define the centers of processing: e.g. Database, Transaction server, logging mechanism etc.
  – Selects connectors which define the modes of interaction. e.g. pipes, events, event multicast, publish-subscribe, RPC, DCOM, etc.
  – Selects Constraints on the architecture e.g. concurrency issues, performance, security etc.
Components, Connectors & Configurations

1. **Components** The centers of processing an architecture. Could be an object, a process, or a function. Independent of a specific system configuration. e.g. Database, a Parser, a Sort Routine, a persistent Parse-tree object, an HTTP client.

2. **Connectors** Creates an interaction between compatible components. Dependent on a specific architecture. e.g., Shared global variable, asynchronous message, RPC, socket connection.

3. **Configuration** a Specific assemblage of components interacting over a set of connectors.
An Architecture (with modeling capability): CORBA

CORBA is a standard for communicating among distributed, heterogeneous systems. These Standards are causing a revolution in software engineering.

- What does it actually do?
- Why distributed? Duh.
- Why Heterogenous:
  1. Different vendors excel at different things;
  2. Legacy systems aren’t going away.
  3. Different languages suit different problems.
- Why a standard To allow implementation heterogeneity: byte order, encoding of arbitrary data (e.g., the HTTP encoding craziness), etc.
- Why an open standard? Promotes competition, allows diversity of implementation choices (trade-offs: CORBA within web browser, CORBA within database server etc).
- Why a revolution
  - A software component market.
  - The possibility of architectural modeling
  - A canned set of architectural styles/design patterns
CORBA Overview

Client object

object Implementation

Proxy Object

CORBA ORB

CORBA ORB
A Modeling Language for components: CORBA IDL

A language for modeling component interfaces (not implementations)

Consider the interfaces between the bidding client and the auction server.

**Auction Server**

```idl
interface bid{ 
    /* struct bitem { 
        String name; 
        Long id; 
    } */
    place(in bitem theBid, 
        out bitem theack) 
    raises ( BidTooLow, 
            AlreadySold );
}
```

**Bidding Client**

```idl
interface clientnotify { 
    oneway void newbid( 
        in String name, 
        in Long id, 
        in Float Amt); 
}
```
What’s the use of CORBA IDL?

1. Models the communication (connector) between the bidding client and the auction server, while ignoring other details. Can be mapped into requirements (use cases)

2. Prescribes/Enforces an interface between the two pieces of software, which are in two different languages (Java and C++). CORBA IDL is language-independent.

3. Promotes information hiding: and therefore reuse, interoperability etc.

4. Supports independent evolution via inheritance (IDL interfaces can be inherited by other interfaces)

5. Actually generates useful code, which takes of architectural ”plumbing” details.
CORBA IDL Code Generation

CORBA IDL Interface

IDL Compiler

Marshalling/UnMarshalling Routines

Java Interface Definitions

Client Program

Marshalling/UnMarshalling Routines

C++ Class Header file

Server Program
Example: A compiler

Perry & Wolf

Two possible configurations: same components, different connectors
Architectural Styles

"An Architectural Style . . . defines a vocabulary of components and connector types, and a set of constraints on how they can be combined . . . there may also exist one or more semantic models that specify how to determine a system’s overall properties from the properties of it’s parts”
—Garlan & Shaw

Describing Architectures

1. vocabulary types of components & connectors.
2. allowable structural patterns
3. underlying computational model
4. invariants of this style.
5. common examples of use.
6. advantages and disadvantages
7. common specializations

Next lecture: Examples of Architectural Styles.