The Architecture Phase of Design

Software Engineering
Computer Science 520/620
Spring 2012
Prof. Leon Osterweil

What is the Nature of Design?

• Addresses the question: HOW?
• Goal: Indicate how to develop a solution system that will satisfy requirements
• Complements:
  – Requirements: WHAT
  – System Test Plan: HOW WOULD I KNOW IT IF I SAW IT
• Design is a very broad and encompassing area
  – Hard to separate it from requirements
  – Hard to separate it from code
• Too hard to be done in one large step
  – Especially because of execution platform variation

Design is essentially a modeling activity

How Does One Go About Designing

• Process by which design is built is understandably complex
• Various authors have differing ideas about this
• For this course, we separate WHAT from HOW

Numerous Well-Known Design Notations and Methods

• Jackson System Development
• Rational Design Method
• Shlaer-Mellor
• Booch Object Oriented Design
• Using various notations
  – DFDs
  – FSAs
  – Petri Nets
  – UML
• ...

These are mostly low-level specification methods

What Do Designs Model (and Why)?

• Conceptual, architectural, high-level designs model how requirements might be met
  – Vehicles for “what-if” discussions
  – Help clarify requirements–by being related to them
  – Often merge and intersperse with requirements
  – Help suggest implementation issues/concerns
• Coding specifications model the form, content, structure of the eventual code
  – Increasing emphasis on evolvability, rapid modification, and flexible deployment
How are Designs Represented?

- Familiar approaches
  - Use of hierarchy to conquer size/complexity
  - Use of multiple views to capture different aspects
  - Use of pictures and diagrams to appeal to non-technical stakeholders
- Connected to requirements elements they respond to
- Connected to code elements that implement them

Architecture vs. Specification

- Architecture
  - High level system design
  - Concerned with components and the interactions among components
  - Not with the algorithms or data structures
- Specification (Low Level Design)
  - Emphasis on data structures and algorithms
  - Focus on implementation issues
    - Stepwise refinement
    - Evolvability
    - Use of abstraction

Traditional Software “Design”

- Early 70’s recognized the need for design
  - Emphasis on data structures and algorithms
    - Stepwise refinement
    - Data abstraction
  - More feasible to think of design as one phase
    - Focused on narrow set of issues
    - Still “Preliminary Design” vs “Detailed Design”
  - Corresponds roughly to what we will be calling “specification”

Architecture addresses new issues, requires a new term

- Change in how we do software development
  - Component based, distributed systems
- Previous HL design notations assumed procedure call model
- Need to focus on
  - Nature of components and their loci
  - Component interactions (Connectors)

Typical Architecture Issues

- Component interaction models
  - What are the components’ interfaces?
  - Who can use them? And how?
  - How much flexibility is achievable? How modifiable?
    - Is plug and play possible?
  - Where is network access used? How?
    - Message passing, broadcasting, etc?
- Late-binding issues
  - Non-determinism
  - Use of proxies

New issues in characterizing system objects

- Interaction protocols
  - Tightly coupled objects
    - Direct or Remote procedure calls
  - Loosely coupled
    - Event based notification, observers
- Degree of separation
  - Locally
  - Internet scale
  - “in the cloud”
- Modes of communicate with each other
  - message passing
  - broadcast
  - multi-cast
“New” Development Approaches
Exploiting ArchitecturePerspectives

- Model-based Development
- Component Based Development
- Service-Oriented Architecture
- Cloud Computing

Model-Based Development

- Start development by creating a model of the system to be built
- Often specified using an architecture definition language
- Use the model to guide development of the system
- Hmmm. Sounds like “design before coding”
  - Think first (?)

Component-Based Development

- Think of development as the composition of pieces (components)
- So, start with a plan for how the pieces will fit together
  - A model?
  - A design?
- Start with some kind of catalog of what pieces are available
  - And what their interfaces are
- Fit the components into their places
- Does that sound familiar?

Service-Oriented Architecture

- Applications composed from components
- Components are accessed via the Web
  - Specified generically (as a “service”)
  - Located by web searches (using proxies)
  - Accessed via the web
- How to compose such services?
  - What composition constructs
  - How to be sure they provide correct services?
  - How to maintain privacy and security?

Service-Oriented Architecture

- SOA approach, but
  - Don’t know/don’t care where or how services are provided via the Web
- Similar problems, but now more worrisome
  - Correctness
  - Security
  - Privacy

Cloud Computing

- This should definitely sound familiar


An Introduction to Software Architecture

The 4 C’s of s/w architecture

• Components
• Connectors
• Constraints
• Configurations

Components and Connectors: The Central Software Architecture Entities

• Components—computational units
  – Subsystems
  – Classes
  – Objects
• Connectors—interaction model
  – Which components are connected to which?
  – How are they connected?
  – Are connectors just components with restricted semantics?

Constraints

• Constraints/attributes
  – Additional information associated with components and connectors
  – Restricts or defines entities

Architecture description (specification or design)

• A high level design that defines the components, connectors, constraints and the inter-relationships among these entities
  – Usually compositional
• The form (i.e. the type of graph) is not the point here
  – What it is trying to communicate (i.e., its position in the development process) is what is important
• Suggests the value of elaborate semantics and annotations of the nodes and edges

Configuration

• A particular instantiation of an architecture description
• Component definitions are represented by actual software systems
  – Conceptual Modules(?)
• Connectors are represented by actual interaction mechanisms (glue code or middleware)
  – CORBA to build connectors
  – Now SOAP
Architectural style

- Restrictions on the architectural specification
- Typically defined using an ADL
- Often captured by a template/pattern
- Examples:
  - Pipe and filter
  - client-server
  - model-view-controller (MVC)

View

- Focuses on some aspect of an architecture
- May filters out some of the information
- May present an alternative representation or visualization of the information

View versus Style

- A specific view can be derived from a given architectural description
  - E.g., layered view
- An architectural description either conforms to a style or it does not

Architecture description language (ADL)

- A language for defining an architecture
- Often has associated capabilities
  - For editing
  - For visualization
  - For analysis
  - For system generation
  - For testing

Some Notable ADLs

- Some examples
  - Darwin
  - Rapide
  - MetaH
  - ACME
  - Menage

References

- Overview:
An Early Precursor to ADLs: Focus on Some Basic Essentials

- Named after Michael Jackson
  - British software design pioneer
- Focus on
  - High levels (with JSD)
  - Lower levels (with JSP)
- Spans from architecture to coding
- More discipline, rigor, focus
- Now overtaken by UML

JSD and JSP

- Jackson System Development
  - Emphasis on high-level conceptual design
  - Develops collection of coordinated graphical depictions of system
  - Strong hints about how to carry them to implementation decisions
  - Strong suggestions about how to go about doing this
- Jackson Structured Programming
  - JSD Based on uses JSP to guide implementation
  - Not covered in this course

JSD Phases

- The modeling phase
  - Entity/action step
  - Entity structure step
- The network phase
  - Connect model processes and functions in a single system specification diagram (SSD)
- The implementation phase
  - Examine the timing constraints of the system
  - Consider possible hardware and software for implementing the system
  - Design a system implementation diagram (SID)

The Modeling Phase: JSD Models

- JSD produces models of the real world and the way in which the system to be built interacts with it
- Primary focus of this is actions (or events)
- Actions can have descriptive attributes
- The set of actions must be organized into set of processes
- Processes describe which actions must be grouped together and what the "legal" sequences of actions are
- Processes can overlap in various ways
- Data are described in the context of actions
  (NOTE: In JSD data considerations are subordinate to actions)
- Processes are aggregated into an overall system model
- Done with the aid of two canonical models of interprocess communication

The Library Example

- Functional requirements:
  - The way books are dealt with
  - The way members are dealt with
- Non Functional requirements
  - Inquiries should be processed as soon as they are received
  - Reports have to be generated at the end of each day.

Models of Processes

- Primary building block of a JSD design
- Contains all actions characterizing a key real-world process
- Actions are structured into a tree
- Only the leaf nodes of the tree are real-world actions
- Interior nodes are conceptual
- Interior nodes can be annotated to show choice or iteration
- Traversals of this tree constitute the only "legal" sequences of actions for this process
- Model a process as a tree: defines a regular expression
- Set of traversals is a regular set
- Process models are usually multiply instantiated
The Book Process

JSD Models: Actions and Entities

- Actions have the following characteristics:
  - An action takes place at a point in time
  - An action must take place in the real world outside of the system.
  - An action is atomic, cannot be divided into subactions.
- Entities have the following characteristics:
  - An entity performs or suffers actions in time.
  - An entity must exist in the real world, and not be a construct of a system that models the real world.
  - An entity must be capable of being regarded as an individual; and, if there are many entities of the same type, of being uniquely named.

Two More Processes

Processes and Data

- Details of actions on data hang off of process model leaf nodes
- Input and Output actions too
  - These are important during the network phase
- Global data is necessary too
  - For functions that must combine data from >1 model process
  - To assure consistency between processes
  - To coordinate between different instances of the same process
  - To coordinate between different instances of the same process

Actions/Attributes

- Acquire
  - attributes: in the library, on loan
- Classify
  - attributes: in the library, requested, time out on loan
- Lend
  - attributes: in the library, on loan, time out on loan, loan date
- Renew
  - attributes: time out on loan, loan date
- Return
  - attributes: in the library, time out on loan, loan date
- Sell
  - attributes: in the library, out of loan
The Network Phase: Communication Between Processes

- Weave Processes together incrementally to form the total system specification
- Also add new processes during this phase: eg. input, output, user interface, data collection
- Goal is to indicate how processes communicate with each other, use each other, are connected to user and outside world
- Linkage through two types of communication:
  --Message passing
  --State vector inspection
- Indicates which data moves between which processes and more about synchronization

Message Passing

State Vector Inspection
Message Passing

- Data stream carries a message from one process activity to an activity in another process (a DFG edge)
  --Must correlate with output leaf of sending model process
  --Must correlate with input leaf of receiving model process
- Data transfer assumed to be asynchronous
  --less restrictive assumption
- No timing constraints are assumed
- Messages are queued in infinitely long queues
- Messages interleaved nondeterministically when multiple streams arrive at same activity

State Vector Inspection

- Modeling mechanism used when one process needs considerable information about another
- State vector includes
  --values of all internal variables
  --execution text pointer
- Process often needs to control when its state vector can be viewed
- Process may need exclusive access to its vector
- Could be modeled as message passing, but --important to underscore characteristic differences
About those Graphs

- There are formal semantics of these graphs
  - Defined by Jackson and colleagues
- Semantics are quite different from the ones we have seen previously

Implementation Phase

- JSD outputs suggest how to proceed with JSP
- Network Phase suggests ideal traversal paths through model processes and their local data
  - Suggests internal implementation of model processes
  - Studying use of model processes suggests internal structure of their data
- Communication by data streams and state vector inspection often suggest real implementations
  - But often not

Comments/Evaluation

- Focus on conceptual design
  - Nobody should assume they will build a system this way
- Based upon model of real world
- Careful (and experienced) analysis of the model generally suggests implementation tactics, though
- Treatment of data is very much subordinated/secondary
- Does a good job of suggesting possible parallelism
- Contrasts strongly with Objected Oriented notions (eg. Booch, UML), that we will see next