Architect’s roles – not just technology

• Creating the “right” technical vision – aligned with organization’s business strategy
• Getting the organization to “buy into” this vision: executive sponsors, project managers, developers – requires a lot of political skill
• Leadership and communication – read Role of the Software Architect for more on this
• We will focus on technical aspects
Software Architecture

• Is a craft, not an engineering discipline, at this point in time (Shaw & Garlan)
• Is a creative, not a routine, activity
• No formal notion of “optimal” architecture, given a product domain
• No repeatable methodology to write down optimal architecture
• Formal training less valuable than experience, including some degree of learning from your own errors (Penny)
However…

- This does not mean you cannot reuse any of the vast experience of architects who came before you
- Can reuse traditional formats and modeling notations for capturing and communicating architectures
- Can reuse architectural styles – popular structures of components and connectors, each having a defined impact on functionality and quality attributes
Format for architecture blueprint

- Introduce domain concepts
- State high-level design goals, principles, constraints
  - to guide detailed design
- Describe system from several viewpoints:
  - Functional (key use cases, sequence of actions performed to realize these use cases)
  - Logical (decomposition into components & connectors, usually object-oriented: tiers, services, packages, possibly classes, and their dependencies)
  - Process (decomposition into processes and threads, choice of communication protocols)
  - Deployment (binding of processes to physical hardware, network structure)
  - Implementation (decomposition of code into layers, choice of API’s)
  - etc.
- Describe required quality attributes and how the architecture enables them
Architecture blueprint should be:

• Self-motivating: include some rationale with your architectural decisions. Don’t leave the reader wondering why you made these choices

• Relevantly biased: not all viewpoints are equally important for all systems – focus on the right aspects (e.g. for an AI system, knowledge base structure & reasoning mechanisms – logical view – deserves more detail than deployment view)

• Simple yet decisive (the hard part)

• Based on known architectural styles
Architectural Styles

• Like patterns in class design
• Common styles (after Shaw & Garlan):
  − Pipes and filters (e.g. Unix shell, data processing)
  − Implicit invocation (e.g. GUI’s: when an event is announced, any interested components may process it)
  − Layered functionality (e.g. network protocols, graphics rendering)
  − Repository / blackboard (e.g. online transaction processing, pattern recognition)
  − Interpreter
More on Architectural Styles

• Attribute-based architectural styles (after Klein & Kazman):
  – Synchronization
  – Data indirection
    • Abstract data repository
    • Publish/Subscribe
  – Layering
  – Simplex

• Attempt to quantify the impact of styles on a system
  – e.g. authors show that Repository reduces the number of code changes in situations where data producers and consumers evolve while data schema stays frozen
Example

• eClaims Exchange – BCE Emergis (2001)
• Requirements highlights:
  – automatically process insurance claims: decide whether to allow or deny, calculate payment, transfer funds
  – support multiple lines of insurance: dental, drug, vision, general health, etc. (even home & auto, if feasible)
  – support all large insurance companies
  – support group or individual insurance
  – insurers maintain up-to-date coverage information and can restructure insurance plans
  – enroll millions of persons
  – support submission of claims via the Web
  – response time under 3 seconds
Architecture highlights – eClaims

• Goals, Principles & Constraints
  – Data producers (enrollment, insurance plan editor, etc.) and consumers (adjudication) must not be designed separately, or they will not work together smoothly. Data schema must be specified before all else.
  – To achieve required extensibility, insurance rules must be externalized (stored as data).
  – Security requirements (access control) can be fulfilled by re-using adjudication functionality (security principals and controlled resources are data entities, and rules can be attached to them already).
Architecture highlights – eClaims

• Relevant styles:
  – Repository (top-level package diagram has star topology)
  – Interpreter

• Use case view: 2 top-level use cases:
  – maintenance
  – adjudication

• Logical view
  – detailed structure of data repository:
    • entities, organized into a hierarchy
    • rules bind to a combination of entities
    • rules have no knowledge of entities they bind to
    • rules are inherited down the hierarchy
    • similar to object-oriented but different
Architecture highlights – eClaims

• Logical view (continued)
  – transactions as protocol for communicating with repository
  – structure of the interpreter:
    • selection of applicable rules
    • execution engine (with rule language specification)
    • reconciling conflicts between rules

• Implementation view:
  – J2EE as platform of choice, Weblogic as application server (corporate standards)
  – Bindings to specific API’s for:
    • persistence
    • transaction management
    • distributed objects
Architecture highlights – eClaims

• Data view:
  – Oracle 8i as database of choice (largely a political decision)
  – mapping of repository structures to relational database tables
  – data access layer on top of JDBC