

What Contributes to an Architectural Design Document

Wendy Liu © 2003

(Acknowledgement: part of the content is contributed by Peter Kanareitsev)

Recall: 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*

Format for architecture blueprint

- Introduce domain concepts
- State high-level design goals, principles, constraints
 - to guide detailed design
- Describe system from several viewpoints (more on the next slide)
 - Functional
 - Logical (& Data)
 - Process
 - Deployment
 - Implementation
- Describe required quality attributes and how the architecture enables them
- Key issues

Model System Using Viewpoints

- Functional and Dynamic
 - key use cases, sequence of actions performed to realize these use cases
- Logical and Structural (& Data)
 - decomposition into components & connectors, usually object-oriented: tiers, services, packages, possibly classes, and their dependencies
- Execution
 - 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

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: JDBC
 - transaction management: JTA
 - distributed objects: RMI

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