Architectural Blueprint

“The 4+1 View Model of Software Architecture”

by Philippe Kruchten

CS 446/646 ECE452
May 30th, 2011
Architectural Model

Definition[1]

- Software architecture = \{\text{Elements, Forms, Constraints}\}

Definition [2]

- “deals with the design and implementation of the high-level structure of the software.

It is the result of assembling a certain number of architectural elements in some well-chosen forms to satisfy the major functionality and .... nonfunctional requirements”


Is this an Architectural Model?

Software/System stakeholders Model

What is going on here?
Desired Attributes

Addresses & captures

- concerns of various *stakeholders*
  - stakeholders:
    - end-users, developers, system engineers, project management
    - testers, support teams
- requirements
  - functional
  - non-functional
    - performance, availability, concurrency, distribution, fault tolerance, security, testing, usability, configuration management, evolution, monitoring
Desired Attributes

An abstraction

- represents the high level view

Is robust

- adaptable
- scalable
- iterative

Meaningful & maintainable

- has to be a live document
  - changes with the system
Types of Architectural Styles

Box & line model

Architectural definition language (ADL)

View based models

- 4+1 view model (1995)
4+1 View Model

Model
- a model is *composed* of 5 *views*
  - a single view is not enough

View
- is catered for a set of *corresponding* stakeholders
  - addresses the concerns of its stakeholders
- contains view elements
  - components, connectors, notation
- generic representation
Perhaps it should have been called 1+4 View Model
Logical View

Intent

- 'object model' of the design
- is generally the starting point
- addresses primarily functional requirements
- decomposition into 'architectural entities'

Style

- abstract data types / OO

Stakeholders

- end-users, architects, designers
Logical View

View representations

• 1. OOA (object oriented analysis)
  – entities are analysis classes
  – application of OOA principles
    • abstraction, encapsulation, inheritance
    • association (aggregation, composition)
  – class diagrams, state diagrams

• 2. data centric analysis
  – entity relationship (ER) diagrams

which is the correct view representation?
Example Logical View

- Conversation
- Translation Services
- Terminal
- Connection Services
- Controller
- Numbering Plan

Pattern Elements:
- Class
- Class Utility
- Parameterized Class
- Class category
- Association
- Containment, Aggregation
- Usage
- Inheritance
- Instanciation
Logical View

Design guidelines

- a **single (object) model** across the system (**why?**)
- avoid premature specialization (**of what?**)
- UML diagrams
  - class, communication, sequence diagrams
Process View

Intent

• handles the **non-functional requirements**
• abstraction of architectural processes
Process View

Architectural Process

- grouping of tasks into executable units
  - task: thread of control
- task hierarchy: major & minor tasks
  - reflects task scope
- types: atomic & distributed
- can be replicated
  - to improve performance, availability etc.
- execute on 'process nodes' (what is a process node?)
Process View

Communication

- messaging (synchronous, asynchronous, RPC, broadcast)
  - usually for major tasks
- shared memory
  - for minor tasks
- can we estimate the system load form the inter-process communication?
Process View

Style

- several styles are applicable
  - pipes & filters
  - layered
    - client / server

Stakeholders

- integrators, architects
Example Process View

Controller process

Terminal process

Main controller task

Controller task
Low rate

Controller task
High rate

Process

Simplified Process

Periodic process adornment

Unspecified

Message

Remote Procedure Call

Message, bidirectional

Event broadcast
Development View

Intent

- software/system decomposition into *software modules*
- software modules
  - name space, packages, libraries, subsystems
  - modules are scoped for small (development) teams

Driven by internal requirements
Development View

Intent

- software/system decomposition into *software modules*
- software modules
  - name space, packages, libraries, subsystems
  - modules are scoped for small (development) teams

Driven by internal requirements

<table>
<thead>
<tr>
<th>management</th>
<th>technology</th>
<th>resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost</td>
<td>requirement allocation</td>
<td>progress monitoring</td>
</tr>
<tr>
<td>evaluation</td>
<td>reuse</td>
<td>&lt;from class&gt;</td>
</tr>
</tbody>
</table>
Development View

Style

- layered style
  - each layer with **well defined** interface
  - subsystem dependencies on other subsystems
    - in the same layer or lower
  - each layer provides a development abstraction (**responsibility**)

Stakeholders

- managers, architects, designers, developers, testers
**Distributed Virtual Machine**

**ATC Framework**

<table>
<thead>
<tr>
<th>Layer</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic elements</td>
<td>Common utilities</td>
<td>Bindings</td>
<td>Low-level services</td>
<td>Aeronautical classes</td>
<td>ATC classes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ATC Functional areas: Flight management, Sector Management, etc.</td>
<td>Man-Machine Interface External systems Off-line tools Test harnesses</td>
</tr>
<tr>
<td><strong>HATS Components</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CAATS, MAATS, etc...</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Hardware, OS, COTS**
Observations

- “complete development architecture can only be described when all the elements of the software have been identified.” [1]

- So what things can we define here?

Development View

Observations

- “complete development architecture can only be described when all the elements of the software have been identified.” [1]

- So what things can we define here?

  code partitioning  module visibility  work allocation
  inter-partition dependencies  development methodology

Physical View

Intent

• physical manifestation of process view
  – processes are mapped to processing nodes

Concerns

• installation, configuration, deployment & delivery, networking, messaging protocols

Stakeholders

• system engineers, installers, architects, operators
Physical View

Design guidelines

- mapping to be flexible
- minimal impact on source code
- same concerns as process view
- UML deployment diagram
Scenario View

Intent

- “one view to rule them all”
- capture system functionality in scenarios
  - interaction of objects & processes
  - driven by important scenarios
- provides architecture validation

Stakeholders

- all stakeholders from the other views
Example Scenario View

Components from the logical view
Connectors from the process view

looks like a collaboration diagrams.
what happened to use case diagram?
Example Scenario View

- **Components** from the logical view
- **Connectors** from the process view

Looks like a collaboration diagrams. What happened to use case diagram?

**use cases & use case realization**
Logical to Process View

Objects are mapped to processes

- considerations
  - autonomy
  - persistence
  - subordination
  - distribution

Strategy

- inside-out: identify processes for objects
- outside-in: identify processes (based on system requests) and then allocate objects to these processes
Logical to Development View

Architectural component decomposition

- architectural entities are broken down into design components
  - packages, modules
  - classes
- mapping is governed by development concerns
- 'distance' between logical and design view
  - an indication of the size of the system
Process to Physical View

Processes assignment to hardware

- major and minor tasks are assigned to physical machines
- various configurations
  - development
  - testing
  - deployment
Model an Iterative Process

Start with a model

In each iteration the architecture is

- prototyped
- tested: under load if possible
- measured & analyzed
- refined
  - add more scenarios
  - detect abstractions and optimizations
- goal:
  - each iteration should take us a step closer to a stable architecture
Comments

Lacks some fundamental views

- security, user interface, testing
- upgrade, disaster recovery

Are the views ever complete?

Change in architectural style?

- data centric to OO architecture