Software Architecture

Lecture 3
Architectural Views and Styles

Rob Pettit
George Mason University
outline

architectural views
  module viewtype
  component & connector viewtype
  allocation viewtype
styles
one system, many views

- a view is a representation of a set of system elements and the relations among them
- not all system elements
- a view selects element types and relation types of interest, and shows only those

why?
one system, many views

- an architect examines the system in three ways
  - how is it structured as a set of code units?
    - module viewtype
  - how is it structured as a set of elements that have run-time behavior and interactions?
    - component & connector viewtype
  - how does it relate to non-software structures, such as hardware and development teams?
    - allocation viewtype
more commonly - 4+1 views

- Adapted from Philippe Kruchen, IEEE Software 12(6)
module viewtype
describes the code structure

- **elements** are **modules**
  code unit that implements a set of functionalities

- **relations** among modules include
  - **A is part of B**
    defines a part-whole relation
  - **A depends on B**
    defines a functional dependency relation
  - **A is a B**
    defines specialization and generalization
different notations exist for module views

- UML class diagrams:

```
    +-----+   +-----+   +-----+
   | A   | -> | C   | -> | E   |
    +-----+   +-----+   +-----+
    ^       |       ^       |
    |       |       | is a |
    `-------'       `-------'
    +-----+   +-----+   +-----+
   | B   |     | D   |     | F   |
    +-----+     +-----+     +-----+
```

- informal: stacked boxes, box-and-line...

examples in a moment
module viewtype
used for code construction and budgeting

construction
- module views are the blueprints for the code
- modules are assigned to teams for implementation
- modules are often the unit for refining the design (e.g., module interfaces)

analysis
- traceability and impact analysis
- budgeting, project management: planning and tracking
module and C&C show different aspects

example program:
- produce alternating case of characters in a stream

module view

C&C view
C&C viewtype describes how the system works

- **elements**
  - components (boxes)
    - principal units of run-time computation and data stores
  - connectors (lines)
    - interaction mechanisms – identity and behavior of their own

- **relations**
  - attachment of components to connectors

- **properties**
  - information for construction & analysis
    - quality attributes
    - others, depending on style (more in a moment)
different notations exist for C&C views

- **ACME diagrams:**

- other notations (normally box-and-line)

examples in a moment
C&C viewtype
used for behavior and QoS analysis

- construction
  - how the system will appear at run time
  - what kind of behavior must be built in
  - pathways of interaction and communication mechanisms

- analysis of runtime properties
  - availability
  - performance
  - security
  - reliability...
allocation viewtype

elements
- software elements
  as defined in module or C&C views
- environment elements
  such as hardware and development teams

relations
- allocated-to
notations for allocation views depend on the **style**

- normally informal, style-specific notations

![Diagram showing the relationship between computing platform, configuration management, deployment style, implementation style, work assignment style, and development organization.](image-url)

*examples in a moment*
outline

architectural views

overview of the first half semester

module viewtype

component & connector viewtype

allocation viewtype

styles
architectural styles:
specialization of element and relation types

- within each viewtype, recurring forms have been widely observed in different systems
- these forms are worth capturing because they have known properties and can be re-used: “tools” in the architect’s “bag of tricks”

an architectural style
is a specialization of element and relation types
together with a set of constraints on how they can be used

- styles exist independently of any system
- two different systems can use the same style
- different parts of the same system may use different styles
remember

- **viewtypes** reflect the three broad ways an architect looks at a system:
  - units of implementation (**module** viewtype)
  - run-time units (**C&C** viewtype)
  - relation to non-software structures (**allocation** viewtype)

- within a viewtype, many choices remain:
  - what kinds of elements are allowed
  - how they relate to each other
  - how are they used or configured

**different styles** result from making different choices
three major styles in the module viewtype

- **decomposition style**
  - hierarchical decomposition of modules
  - supports concurrent development

- **generalization style**
  - specialization hierarchy
  - supports reuse; managing large numbers of definitions

- **layered style**
  - virtual machines
  - supports portability, reuse
decomposition style
in the module view type

- elements are modules
- relations restricted to $A$ is part of $B$

what it is for

- a starting point
  - frequently, assigning functions to modules
  - is a prelude to detailed design
- change/impact analysis
- basis for work assignments
  - provides elements in the allocation view
decomposition style in the module viewtype

examples in UML
decomposition style in the module view type

- outline/tree examples

<table>
<thead>
<tr>
<th>Software Decision Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Data Type Module</td>
</tr>
<tr>
<td>Numeric Data Type Module</td>
</tr>
<tr>
<td>State Transition Event Mod.</td>
</tr>
<tr>
<td>Data Banker Module</td>
</tr>
<tr>
<td>Singular Values Module</td>
</tr>
<tr>
<td>Complex Event Module</td>
</tr>
<tr>
<td>Filter Behavior Module</td>
</tr>
<tr>
<td>Physical Models Module</td>
</tr>
<tr>
<td>Aircraft Motion Module</td>
</tr>
<tr>
<td>Earth Characteristics Module</td>
</tr>
<tr>
<td>Human Factors Module</td>
</tr>
<tr>
<td>Target Behavior Module</td>
</tr>
<tr>
<td>Weapon Behavior Module</td>
</tr>
<tr>
<td>Software Utility Module</td>
</tr>
<tr>
<td>Power-Up Initialization Module</td>
</tr>
<tr>
<td>Numerical Algorithms Module</td>
</tr>
<tr>
<td>System Generation Module</td>
</tr>
<tr>
<td>System Generation Parameter Mod.</td>
</tr>
<tr>
<td>Support Software Module</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Behavior-Hiding Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Driver Module</td>
</tr>
<tr>
<td>Air Data Computer Module</td>
</tr>
<tr>
<td>Audible Signal Module</td>
</tr>
<tr>
<td>Computer Fail Signal Module</td>
</tr>
<tr>
<td>Doppler Radar Module</td>
</tr>
<tr>
<td>Flight Information Display Module</td>
</tr>
<tr>
<td>Forward Looking Radar Module</td>
</tr>
<tr>
<td>Head-Up Display Module</td>
</tr>
<tr>
<td>Inertial Measurement Set Module</td>
</tr>
<tr>
<td>Panel Module</td>
</tr>
<tr>
<td>Projected Map Display Set Module</td>
</tr>
<tr>
<td>Shipboard Inertial Nav. Sys. Mod.</td>
</tr>
<tr>
<td>Visual Indicator Module</td>
</tr>
<tr>
<td>Weapon Release Module</td>
</tr>
<tr>
<td>Ground Test Module</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shared Services Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode Determination Module</td>
</tr>
<tr>
<td>Panel I/O Support Module</td>
</tr>
<tr>
<td>Shared Subroutine Module</td>
</tr>
<tr>
<td>Stage Director Module</td>
</tr>
<tr>
<td>System Value Module</td>
</tr>
</tbody>
</table>
generalization style in the module viewtype

- elements are modules
- relations restricted to $A$ is a $B$
- properties
  - inheritance semantics: interface vs. implementation

what it is for
- basis for object-oriented designs
- supports evolution and extension
- reuse
generalization **style** in the module viewtype

- examples in UML

![Diagram showing generalization style](image)

- reflected in programming languages
  - **Circle** extends **Shape**
layered style
in the module viewtype

- elements are layer modules
- relations restricted to A allowed to use B
  a special case of A depends on B

- stylistic rules
  - every piece of software is assigned to exactly one layer
  - software in a layer is allowed to use software in
    {any lower layer, next lower layer}
  - software in a layer {is, is not}
    allowed to use other software in same layer

- what it is for
  - separation of concerns/incremental development
  - portability

style variations:
layered style in the module viewtype

- examples (interpret each one)

- stack of boxes

  - A
  - B1  B2  B3
  - C

- boxes and arrows

- concentric rings

A

B1  B2  B3

C

D
layered style in the module viewtype

example: Google Android's Architecture

is this a good description?
(interpret it according to the style variations)
many styles
in the C&C viewtype

data flow
  batch sequential
  dataflow network (pipe & filter)
    acyclic, fan-out, pipeline, Unix
  closed loop control

call-and-return
  main program/subroutines
  information hiding – objects
  stateless client-server
  SOA

interacting processes
  communicating processes
  event systems
    implicit invocation
  publish-subscribe

data-oriented repository
  transactional databases
  stateful client-server
  blackboard
  modern compiler

data-sharing
  compound documents
  hypertext
  Fortran COMMON
  LW processes

hierarchical
  tiers
    interpreter
  N-tiered client-server
pipe & filter style in the C&C viewtype

- elements are pipes (data flow) and filters (computation)
- relations restricted to P.in/out attached to F.port
- what it is for
  - functionality related to data streaming and transformation
    e.g. media streaming, image processing,…
event publish-subscribe style in the C&C viewtype

- elements are objects/threads and events
- relations restricted to A publishes E, A subscribes E

- two style variants
  - implicit invocation: one responder will be passed the event
  - publish-subscribe: zero or many subscribers (no guaranties)

- what it is for
  - high degree of separation between functional units
    e.g. Google Android
three major styles
in the allocation viewtype

- **deployment style**
  - allocates software elements, i.e. code, to processing and communication nodes
  - properties include those necessary to calculate (and achieve) performance, availability

- **implementation style**
  - allocates software elements to structures in the development environment’s file systems
  - properties include files and capacities

- **work assignment style**
  - allocates software elements to organizational work units
  - properties include skill sets
deployment style
in the allocation viewtype
example: informal notation
in Summary

- **views** help manage the complexity of describing an architecture

- **viewtypes**
  determine the kinds of things a view talks about
  - three primary viewtypes: module, C&C, allocation

- each viewtype has many **styles**
  - module: decomposition, generalization, layered, ...
  - C&C: pipe & filter, client-server, pub-sub...
  - allocation: deployment, work assignment...