Architecture Description Languages (ADLs):
Introduction, Koala, UML as an ADL

T-76.150 Software Architecture
Timo Asikainen

Contents

- Brief motivation for ADLs
- General features of ADLs
- Koala
- UML as an ADL
- Summary
**Motivation for ADLs**

- Explicit description of architecture needed
- Why another modelling method/set of methods?
  - Natural language does not work
  - Informal box-and-line diagrams too informal and lack expressive power
  - No existing method (e.g., UML) is perfectly suited for describing architecture
  - Idioms are not able to capture all relevant architectural designs

**General Features of ADLs**

- Components (in all ADLs)
- Connectors (strong emphasis)
- Systems comprised of components and connectors
- Both textual and graphical syntax
- Formal syntax and semantics
- Internal consistency rules: what is legal and what is not?
General Features of ADLs (cont’d)

- Modelling both individual systems and families of systems or architectural styles
- Hierarchical levels of design
- Behaviour modelling
- Little empirical evidence of usefulness or reports from industrial use

Architecture description languages (ADLs)

- ACME
  - Components and connectors
- Wright
  - Components and connectors, behaviour
- Koala
  - In use at Philips
  - Support for product families
- UML
  - Controversial opinions
  - In practice, useful — and used
Acme – The Core Concepts of ADLs

- Acme was originally designed as a joint effort of the architecture community to serve as an architectural interchange language.
- Therefore, Acme were given the concepts that are shared by all the ADLs.

**Concepts**

- Component, connector
- Port, role, attachment
- Property
- System
- Representation, binding
- Style

Acme Concepts – Component, Connector

- Component
  - Loci of computation
  - Data storages
  - Examples: server, client, filter

- Connector
  - Interactions between components
  - Examples: procedure call, pipe, HTTP-protocol
Acme Concepts – Port, Role, Attachment

- Port and role
  - Intensional points of interactions in components and connectors, respectively
- Attachment
  - Connect ports and roles in component and connectors
  - No other connections between components and connectors
- Property
  - Represent ‘other information’ about entities

Acme Concepts – Systems, Representations

- System
  - Set of components attached together through connectors (configuration of components and connectors)
- Representations
  - Components and connectors can be given one or multiple representations
  - Bindings define relations between ports and roles in representing and represented components
Acme Concepts – Style

- A style defines a design vocabulary in terms of:
  - component, connector, port, and role types
  - constraints that systems in the style must satisfy

Examples

- Pipe-and-filter style: all components are filters, all connectors are pipes

---

A Sample Acme System

```
System simple_cs = {
    Component client = { Port send-request; };
    Component server = { Port rec-request; }
    Representation rep = {
        System sub = {
            Component c = { Port p; }
        }
        Bindings {
            server.rec_req to server.rep.sub.c.p;
        }
    }
    Connector rpc = { Roles { caller, callee };
        Attachments {
            client1.send_req to rpc.caller;
            client2.send_req to rpc.caller;
            server.rec_req to rpc.caller;
        }
    }
};
```
Wright

- Basically, all the important concepts of Acme are included in Wright as well.
- In addition, Wright models the behaviour of components and connectors using a formalism called CSP (Communicating Sequential Processes).
  - This enables the analysis of the properties of systems composed from components and connectors.
    - Compatibility of ports and roles, dead-locks etc.

Koala

Picture from Rob van Ommering
Koala - Background

- Designed at Philips Consumer Electronics
- Used in developing embedded software for consumer electronic products, e.g. television sets, DVD-players etc.
- Koala involves an architecture description language based on Darwin, an earlier ADL
- Additionally, Koala includes support tools, process guidelines, and methods for managing the assets
- This lecture will concentrate on the ADL part of Koala

Koala – Motivation

- Practical relevance: one of the very few reported uses of an ADL in the industry
- Interesting concepts for representing software architecture
Koala – Business Case

- Each embedded system developed is burned on a ROM and installed in a large number of units
- Bugs are nasty: consider updating the chip in 10,000 television sets spread around the world
- Speed and size are critical: computation power and memory are not in ample supply

Koala – Basic Concepts – Component

- Units of design, development, and reuse
- The main design element
- Each component instance has a single type
- Components can contain other components
- Components defines its communications possibilities (interface) through a set of interface instances
Koala – Basic Concepts – Interface Type

- A set of semantically related functions
  - Compare with interfaces in Java, COM, or UML
  - Functions are specified through standard function signatures – compare with function definitions in C

```java
interface Irpc {
    void MakeCall(int n);
}
```
Note – Different Meanings of ‘Interface’

1. The entire set of possibilities which an entity has for interacting with other entities
2. A well-defined point of interaction of an entity
3. A small set of semantically related operations

Koala – Provided and Required Interfaces

- Components define interfaces
- Each interface definition includes the following
  - The direction of the interface: either provided or required
  - Interface type
  - Name of the interface
Koala – Interface Binding Rules

- Interfaces can be bound together
- However, there are some rules for the bindings
- In graphical terms, the tip of an interface must be bound to a base of exactly one other interface
  - In other terms, all function calls must have exactly one target
  - All provided services need not be used
Koala – Hierarchical Structure

- Hierarchical structure is defined in terms of components contained within other components
- Each contained component definition includes
  - Component type
  - Component name
- Above-mentioned binding rules apply to interfaces in contained and containing components as well
Koala – Binding Semantics

- In the basic form of binding between interfaces, calls to functions in required interfaces are textually replaced with a call to the function with the same name in the provided interface.
- More complex connections can be defined in modules.
  - Connecting individual functions, wrappings.
  - In a sense, modules are the connectors of Koala.
  - However, no strong focus on connectors.

\[
\text{#define } r_f \quad \text{c_p_f} \quad \text{#define } p_f \quad \text{c_p_f}
\]
Koala, example

UML as an ADL
### Fundamentals – Why to Use UML?

- UML is the de-facto standard of software modelling
  - The only method with considerable tool support
  - The only widely-spread method
- Software engineers do not have the time or interest to learn new modelling methods, especially formal ones
- UML provides a wide range of different concepts that can potentially be exploited in modelling architecture

### Architecture Modelling with UML 2.0

- Compared to UML 1.x, UML 2.0 provides good facilities for modelling architectures
- UML 2.0 is to be released “sometime around the end of 2000”
- In earlier versions of UML, it was not obvious which concepts should be used to capture architectural concepts
- With UML 2.0, components are an obvious choice to model components
- There are concepts for modelling the connection points are well...
Component Diagrams

- Compared to UML 1.x, components are given extra emphasis in UML 2.0
  - In UML 1.x, components are related to implementation
  - In UML 2.0, a more holistic treatment is given
- Most important additions
  - Internal structure of components in terms of its parts
  - Connectors between parts
  - Provided and required interfaces that may be aggregated into ports

Characteristics of Component

- A component is a self contained unit that encapsulates the state and behavior of a number of classifiers
- A component is a substitutable unit that can be replaced at design time or run-time by a component that offers equivalent functionality based on compatibility of its interfaces
- A component can only be accessed through its provided interfaces
Standard Notation for Classifiers

- Standard notation for classifiers (i.e., similar to the one used in class diagrams) may be used with components as well

- As an example, the functions contained in an interface can be represented in a drawing using this alternative
Parts and Connectors

- Components may have both classes and other components as their parts
- Connectors are a special form of associations
- Two kinds of connectors
  - Assembly
    - Between a required-provided pair
    - Ball-and-socket notation
  - Delegation
    - Between a provided-provided or required-required pair of interfaces or ports
    - Arrow-to-ball notation
  - The same type may be used as the type of several components and connectors

---

White-Box View - An Example
Connectors

- A component may contain multiple interfaces, and is typed by the types and directions (provided, required) of its interfaces.

Ports

- The notation for a port is a rectangle at the boundary of a component.
- A component may contain multiple interfaces, and is typed by the types and directions (provided, required) of its interfaces.
Ports

- "A port is a structural feature of a classifier that specifies a distinct interaction point between that classifier and its environment or between the (behaviour of the) classifier and its internal part." [UML 2 standard]

Uses of Ports

- Connectors need not be attached to ports at both ends
Deployment Diagrams

- Deployed artefacts may be specified as a textual list.
- Deployment specifications may be used to define details: “a set of properties … execution parameters.”
  - Specification
  - Instance

Deployment Specification in a Context

- The folded-paper icon (□) stands for an artifact.
Architecture Modelling with UML 1.x

- In earlier versions of UML, there are a number of possible choices to model architectures
- Individual choices can be combined in different ways
- In the following, we will briefly discuss the different possibilities

Freedom of Choice – Components, Connectors

- UML concepts applicable to modelling components
  - Class
  - Component
  - Package
  - Subsystem
- UML concepts applicable to modelling connectors
  - All the above-mentioned and
  - Association, association class
  - Dependency
Freedom of Choice - Ports

- UML concepts applicable to modelling ports
  - Class
  - Interface
- There is a semantic mismatch between UML interfaces and ADL ports
  - A class can realise an interface at most once, whereas a component can use the same port type many times

Freedom of Choice - Conclusions

- Freedom of choice is nice to have
  - With many options, at least one is likely to work
- However, the absence of a standard way of modelling architecture with UML is the cause of problems
  - Must cope with multiple diagram types
  - There exists no general, standard way how to use UML (1.x) of modelling software architecture
    - Of course, such a way may exists in a certain company, or for a certain problem domain
UML and Politics

- The standardisation process of UML is not driven solely by scientific arguments
  - Lots of politics about what will be taken in and left out
  - Tool vendors affect the process
- UML is far from being a solid basis to build on
- However, unlike most (all) other methods, UML is supported by books, tools, and consultants

Stick to UML

- UML is a language, not mere set of notational symbols
- Compare with natural language: Are these English?
  - The owl sits in the tree.
  - A owl sit tree in.
  - asdklhfwetm. aewnrhj
- Be careful when using extensions to the standard
  - Many extensions have been suggested, but not many of them are widely-known
ADLs and the Big Picture

- How do ADLs compare to the stuff on the course so far?
- Views
  - A typical ADL describes a single, specific view
- Styles
  - ADLs can be used to enforce specific styles for systems

Conclusions

- At least semi-formal methods are needed for adequately describing software architectures
- A large number of ADLs have been suggested
  - However, there is no agreement on what should be modelled
  - There is no evidence that ADLs are widely adopted in the industry – Koala is the prime example
- UML can be used for modelling architecture, especially its upcoming version (2.0)
Literature

- Acme

- Koala

- ADLs in general