
Architectural Description Languages

R.H. Mak
Goals

• Students should understand the relationship between architectural frameworks and architectural description languages.
• Students should have some basic knowledge of some specific ADLs (in particular UML).
• Students should be able to recognize whether, and if so to what extent, a language classifies as an ADL.
• Students should be able to judge an ADL on its suitability to define architecture frameworks for a specific domain.
Agenda

- Introduction
- ACME
- UML
- AADL
Agenda

• Introduction
• Definition
• Concepts
• Comparison framework
• ACME
• UML
• AADL
Definition: ISO/IEC/IEEE 42010

- An architecture description language (ADL) is any form of expression for use in architecture descriptions.
- An ADL provides one or more model kinds as a means to frame some concerns for its audience of stakeholders.
- An ADL can be narrowly focused, defining a single model kind, or widely focused to provide several model kinds, optionally organized into viewpoints. Often an ADL is supported by automated tools to aid the creation, use and analysis of its models.
The ADL conceptual model is almost identical to the architecture framework conceptual model of an architecture.

By using a specific ADL the architect adopts a specific architecture framework,
  - e.g. uses a specific viewpoint library.
General purpose versus specific

ADLs can be general purpose (widely focused) such as UML, or tailored to a specific application domain or a specific architectural style (narrowly focused). As a rule of thumb:

- Domain specific ADLs provide better support for analysis, predict qualities of an architecture with greater accuracy, allow generation of models for simulation.
  - Specific domains could be real-time embedded systems, avionic, automotive, …
- General purpose ADLs provide broader support for multiple views, models.
- Architectural styles will be treated in a separate lecture.
Remarks

- ISO/IEC/IEEE 42010 definition not very helpful
  - Too vague
- Acme (Garlan, Monroe, Wile)
  - Present core ontology (fundamental concepts and their relationships)
- Paper by Medvidovic Taylor (2000)
  - Starts from Acme core ontology
  - Defines a comparison framework:
    - Criteria by which to judge concepts from the ontology
  - Classifies languages that are commonly perceived as ADLs
    - Out-dated, only languages up to 2000
ACME: core ontology

- Components
- Connectors
- Systems
  - Configurations of components and connectors
- Ports
  - Points of interaction with a component
- Roles
  - Points of interaction with a connector
- Bindings (rep-map in ACME parlance)
  - Mappings from a composite component’s (or connector’s) internal structure to elements of its external interface
Medvidovic-Taylor Framework

- An ADL is a language that provides features for *modeling* a software system’s conceptual architecture.
  - it provides concrete syntax
  - It defines a conceptual framework
    - typically domain specific

- Comparison w.r.t.
  - Modeling features for (subset of the ACME ontology)
    - Components
    - Connectors
    - Configurations
  - Tool support
Comp-Conn modeling features (1)

- **Interface**
  - **adamant** for components
  - fully specifies all possible interaction

- **Types**
  - encapsulate functionality, facilitate reuse through repeated instantiation (within and across architectures)

- **Semantics**
  - High level model of behavior
    - To perform analysis and enforce constraints
    - To establish consistency of ADs at different levels of abstraction
    - For connectors given by protocols
Comp-Conn modeling features (2)

- Constraints
  - Properties and assertions
    - To ensure adherence to intended use
- Evolution
  - Features that facilitate future modification due changing requirements on the system
    - Subtyping, refinement
- Non-functional (extra-functional) properties
  - Needed for prediction of runtime behavior, performance, deployment on runtime platform, …
Configuration modeling features (1)

- Understandability
  - Simple syntax for describing the topology
  - Important for communication with stakeholders
- Compositionality
  - Mechanisms for creating hierarchies of subsystems
- Refinement / traceability
  - About correspondence between ADs (models) on different levels of abstraction.
- Heterogeneity
  - Ability to cope with components and connectors developed in different languages, for different OS, …
Configuration modeling features (2)

• Scalability
  • hampered by fixed numbers (interface/component, ports/connector, etc…), support for enumeration helps

• Evolvability
  • Features that support extending and modifying the architecture through new versions of the system
  • Important for maintenance

• Dynamism
  • Possibility to modeling runtime modifications to the architecture
  • Important for resource management and reliability
Configuration modeling features (3)

- Constraints
  - Global ones, e.g. each connector connects a required interface to a provided interface
  - separate constraint language

- Non-functional properties
  - Mechanisms to compute/verify the global property from the local ones, e.g. a system is available if all its components are running, latency of a pipeline is the sum the latency of its stages, etc
Tool support features

- Active specification
  - Options for specification vary depending on already specified parts
- Multiple views
  - In particular preserving consistency between views
- Analysis
  - E.g. of performance, simulation, constraint checking
- Refinement
- Implementation generation
- Dynamism
  - Support for dynamic loading and linking
Agenda

• Introduction
• ACME
  • Characterization
  • Concepts
  • Vocabulary
• UML
• AADL
• Original intention:
  • to serve as interchange format for exchanging architectural information between separately-developed ADL tools
  • as a consequence it establishes a common ontology, i.e., a set of fundamental concepts and their relationships

• Secondary goals
  • support for tool development
  • foundation for new domain specific ADLs and standards for architectural descriptions
  • human readable descriptions

• Main focus on structural and quality aspects
  • behavioral aspects through code reference
  • thereby also providing a form of operational semantics
Acme: example
System simple_cs = {
    Component client = {
        Port send-request;
        Properties {
            Aesop-style: style-id = client-server;
            UniCon-style: style-id = cs;
            source-code: external = "CODE-LIB/client.c"
        }
    }

    Component server = {
        Port receive-request;
        Properties {
            idempotence: boolean = true;
            max-concurrent-clients: integer = 1;
            source-code: external = "CODE-LIB/server.c"
        }
    }

    Connector rpc = {
        Roles {caller, callee}
        Properties {
            synchronous: boolean = true;
            max-roles: integer = 2;
            protocol: Wright = "..."
        }

        Attachments {
            client.send-request to rpc.caller;
            server.receive-request to rpc.callee
        }
    }
}
Acme core: 7 types of entities

Plus: Representations and Rep-maps
Agenda

- Introduction
- ACME
- UML
  - Characterization
  - Concepts
  - Vocabulary
- AADL
Unified Modeling Language

- A graphical language
- for visualizing, specifying, constructing and documenting artifacts of a software-intensive system
- originated from unification of methods by Booch, Rumbaugh (OMT) and Jacobson (OOSE)
- standardized by the OMG
- with a well-defined syntax and semantics
  - including the possibility for user-defined extensions
- with support for Kruchten’s 4+1 views
- with tool support for Model-Driven Architecture (MDA)
UML: terms and concepts

- **System**
  - A set of elements organized to accomplish a purpose and described by a set of models possible from different viewpoints
  - Can be decomposed into a set of subsystems

- **Model**
  - A simplification of reality, an abstraction of a system, created in order to better understand the system

- **View**
  - A projection of a model, which is seen from one perspective or vantage point and omits entities that are not relevant to this perspective
UML vocabulary

- Things
  - Structural, behavioral, grouping, annotational
- Relationships
  - Dependencies, associations, generalizations, realizations
- Structural diagrams
  - class diagram, object diagram, component diagram, deployment diagram
- Behavioral diagrams
  - use case diagram, interaction (sequence or collaboration) diagram, statechart diagram, activity diagram
Structural things (1)

- Represent building blocks for modeling the static structure of the architecture
- Dynamic models contain separate building blocks
Structural things (2)

Collaboration

- Chain of responsibility
- extra compartments may be used to show contents

Use Case

- Name
- Place order

Active Class

- Name
- EventManager
- q: EventQueue
- suspend()
- flush()

Component

- Name
- spelling.java
- attributes
- operations
- extra compartments may be used to show contents

Node

- Name
- Server
Relationships

Dependence

- name
- source: Owner
- target

Generalization

- discriminator
- child: power
- parent

Association

- name
- multiplicity
- navigation
- end
- interface specifier
- role name
- unfilled diamond for aggregation
- filled diamond for composition
Behavioral things (1)

- Entire model (sequence diagram): contains both building blocks and relationships
Behavioral things (2)
Diagrams (Model kinds)

- Structural diagrams
  - Class diagram
  - Object diagram
  - Component diagram
  - Deployment diagram

- Behavioral diagrams
  - Use case diagram
  - Interaction diagram (sequence & collaboration)
  - State machine diagram
  - Activity diagram

This is not all: see book by Booch, Rumbaugh, Jacobson, or tutorials on the WWW
UML version: Kruchten 4+1 view
Remarks

• No real consensus on definition of an ADL
  • E.g., Medvidovic and Taylor do not consider UML to be an ADL because
    • Connectors are not first class citizens
    • Its focus is not primarily on conceptual architecture
  • UML + … is usually acceptable
    • E.g. + Object Constraint Language (OCL)
    • Make use of UML extensions
    • SysML an extended subset of UML for Systems Engineering
      • A practical guide to SysML
      • You have electronic access to this book via the TU/e library
Agenda

• Introduction
• Overview
• UML
• AADL
  • Characterization
  • Concepts
  • Vocabulary and notations
  • Example
“The AADL is a unifying framework for model-based software systems engineering that you use to capture the static modular software architecture, the runtime architecture in terms of communicating tasks, the computer platform architecture on which the software is deployed, and any physical system or environment with which the system interacts.”

Peter H. Feiler, David P. Gluch

Model-Based Engineering with AADL
AADL characteristics

- Is a SAE aerospace standard (AS5506)
- Employs formal modeling concepts
- Has a well-defined semantics
- Uses both textual (2) and graphical notation
- Has tool support (OSATE, ...)
- Supports model-based analysis
- Focusses on performance-critical properties of real-time embedded and high-dependability systems
AADL component abstractions

Divided into three subcategories

- Application software
  - Thread, Thread group, Process, Data, Subprogram
- Execution platform
  - Processor, Memory, Device, Bus
- Composite
  - System
AADL runtime semantics

For data exchange and control mechanisms such as:

- Message passing
- Event passing
- Synchronized access to shared components
- Thread scheduling protocols
- Timing requirements
- Remote procedure calls
AADL interfaces

AADL components interact exclusively through defined interfaces:

Directional flows through:

- Data ports, for unqueued state data
- Event data ports, for queued message data
- Event ports, for asynchronous events
- Synchronous subprogram calls
- Explicit access to data components
AADL’s three model representations

AADL Textual

thread data_processing
features
raw_speed_in: in data port:
speed_out: out data port;
Properties
Period => 20 ms;
end data_processing;

AADL Graphical

20 ms
data_processing

XML

<threadType name="data_processing"> 
<features>
<dataPort name="raw_speed_in"/>
<dataPort name="speed_out"
direction="out"/>
</features>
</threadType>
AADL graphical notation

Component abstractions

Interfaces

Application Software

- data
- process
- thread group
- thread
- subprogram

Execution Platform

- device
- memory
- bus
- processor

Composite

- system

package

Data port

in

in out

out

Event port

Event data port
process control_processing
features
input: in data port;
output: out data port;
end control_processing;

process implementation control_processing.speed_control
subcomponents
control_input: thread control_in.input_processing_01;
control_output: thread control_out.output_processing_01;
control_thread_group: thread group
control_threads.control_thread_set_01;
set_point_data: data set_point_data_type;
end control_processing.speed_control;

thread control_in
end control_in;

thread implementation control_in.input_processing_01
end control_in.input_processing_01;

thread control_out
end control_out;

thread implementation control_out.output_processing_01
end control_out.output_processing_01;

thread group control_threads
end control_threads;

thread group implementation control_threads.control_thread_set_01
end control_threads.control_thread_set_01;

data set_point_data_type
end setpoint_data_type;
system integrated_control
end integrated_control;
--

system implementation integrated_control.integrated_control_system
subcomponents
control_process: process controller.speed_control;
set_point_data: data set_points;
navigation_system: system core_system.navigation;
real_time_processor: processor rt_fast.rt_processor;
hs_memory: memory rt_memory.high_speed;
high_speed_bus: bus network_bus.HSbus;
end integrated_control.integrated_control_system;
thread read_data
features
  in_data: in data port;
  out_data: out data port;
properties
  Period => 50 ms;
end read_data;

thread basic_control
features
  in_data: in data port;
  out_data: out data port;
properties
  Period => 50 ms;
end basic_control;

process implementation control_speed.impl
  subcomponents
    read_data: thread read_data;
    control: thread basic_control;
  connections
    immediate_C1: data port read_data.out_data -> control.in_data;
end control_speed.impl;
system implementation complete.impl
subcomponents
 brake_pedal: device brake_pedal;
 cruise_control: system cruise_control;
 throttle_actuator: device throttle_actuator;
connections
 C1: event data port brake_pedal.brake_event -> cruise_control.brake_event;
 C2: data port cruise_control.throttle_setting -> throttle_actuator.throttle_setting;
flows
 brake_flow: end to end flow brake_pedal.Flow1 -> C1 ->
 cruise_control.brake_flow -> C2 -> throttle_actuator.Flow1;
end complete.impl;

--

device brake_pedal
features
 brake_event: out event data port;
flows
 Flow1: flow source brake_event;
end brake_pedal;

--

system cruise_control
features
 brake_event: in event data port;
 throttle_setting: out data port float_type;
flows
 brake_flow: flow path brake_event -> throttle_setting;
end cruise_control;

--

device throttle_actuator
features
 throttle_setting: in data port float_type;
flows
 Flow1: flow sink throttle_setting;
end throttle_actuator;

--
data float_type
end float_type;
Literature


OMG UML standard and related documents