Architecture
Description
Languages & Tools

Carlos Diego Damasceno
Milena Guessi

SCC-5944 Software Architecture – 2016
Profa. Dra. Elisa Yumi Nakagawa
1. History
2. Architecture modeling elements
3. ISO/IEC/IEEE 42010
4. Formalism levels
5. Examples
6. Tools
7. State-of-the-practice
Traditional Definitions

- Provide mechanisms for expressing composition, abstraction, reusability, configuration, and analysis of software architectures (Shaw and Garlan, 1994)

- An ADL must explicitly model components, connectors, and their configurations; furthermore, to be truly usable and useful, it must provide tool support for architecture-based development and evolution (Medvidovic and Taylor, 2001)
Characteristics

• Architecture building blocks
  o Components
  o Connectors
  o Configurations

• Tool Support
  o Automated analyses on the architecture description
**Architecture modeling elements**

- **Components and Connectors**
  - Interface
  - Type
  - Semantics
  - Constraints
  - Evolution
  - Non-functional properties

- **Tool Support**
  - Active specification
  - Multiple views
  - Analysis
  - Refinement
  - Implementation generation
  - Dynamism

- **(Architectural) Configuration**
  - Understandability
  - Compositionality
  - Refinement and traceability
  - Heterogeneity
  - Scalability
  - Evolution
  - Dynamism
  - Constraints
  - Non-functional properties

An ADL is any form of expression for use in architecture descriptions.

ISO/IEC/IEEE 42010


Conceptual model of an architecture description language
Informal
- Present neither defined syntax or semantics
- Main usage:
  - Illustrating or exemplifying concepts

Semi-formal
- Present defined syntax but lack a complete semantics
- Main usage:
  - Supporting communication among stakeholders

Formal
- Present formally defined syntax and semantics
- Main usage:
  - Verifying and validating models against properties and quality attributes
Examples
• Many, many, many ADLs...
  - 123!!
Examples

Informal

1. Modules can (a) provide interfaces, hiding other modules, or (b) exposing some interfaces of internal modules.

2. A bird’s-eye-view of a system as it appears at run-time.

3. Shared data view of an agent

Source:
1,2 Clements, P. et al. Documenting Software Architectures: Views and Beyond. Addison-Wesley, 2011
Examples
Semi-formal

1. UML 2.x diagram types

2. SysML 1.x diagram types

Source:
1 http://www.omg.org/spec/UML/2.5/
2 http://www.omg.org/spec/SysML/1.4/
Examples

Semi-formal UML

Examples

Semi-formal SysML

Source: http://www.omgsysml.org/
Examples
Semi-formal
SysML

Source: http://www.omg.sysml.org/
**Examples**

**Formal**

A composite component specified in *Darwin* (top) and (bottom) the graphical view of the component

```
component Composite
provide provserv;
require reqserv;
inst
  C1 : CompType1;
  C2 : CompType2;
bind
  provserv -- C1.pserv;
  C2.rserv -- reqserv;
```

**Dynamism**

Dynamic insertion of a component into a *C2SADEL* architecture.

**Style Pipe-Filter**

**Constraints**

\[
\forall c : \text{Connectors} \quad \text{Type}(c) = \text{Pipe}
\]

\[
\exists \forall c : \text{Components} \quad p : \text{Port} \quad p \in \text{Ports}(c) \quad \text{Type}(p) = \text{Data Input} \quad \lor \quad \text{Type}(p) = \text{Data Output}
\]

The pipes-and-filters style declared in *Wright*.

**Evolvability**

```
Family fam = {
  Component Type comp1 = { Port pl; }
  Component Type comp2 = { Port pl; }
  Connector Type conn1 = { Roles pl; }
}

Family sub_fam extends fam with {
  Component Type sub_comp1 extends comp1 with {
    Port pl = { Property attach : int <<default = 1>>; }
  }
  Component Type comp3 = { ... }
}
```

Declaration in *ACME* of a family of architectures, *fam*, and its subfamily, *sub_fam*, which has new components and properties.

Examples

Formal

\(\pi\)-ADL

Description of a simple pipeline architecture

Source: Cavalcante, E., Oquendo, F., Batista, T. Architecture-Based Code Generation: From \(\pi\)-ADL Architecture Descriptions to Implementations in the Go Language. ECSA 2015.
Tools
Tools

PolarSys

• Description:
  o Eclipse-based solution for SysML and UML modeling

• Features:
  o Model-based simulation, formal testing, safety analysis, performance/trade-offs analysis, architecture exploration
  o Free and open source

• Support:
  o UML
  o SysML
  o ISO/IEC 42010

• Homepage:
  o https://www.polarsys.org/
Tools

Enterprise Architect

• Description:
  o High performance modeling, visualization, and design platform based on the UML 2.5

• Features:
  o Business Modeling, Requirements Traceability, Document Generation, Source Code Generation, Reverse Engineering, Systems Engineering and Simulation
  o Trial version (Academic price)

• Support:
  o UML 2.5
  o BPMN
  o SysML
  o MDA
  o C/C++
  o Java

• Homepage:
Tools

ArchiMetric

• Description:
  o An all-in-one software and system development tool for end-to-end IT system modeling

• Features:
  o Enterprise Modeling, Document Production, Project Management
  o Full-featured trial for 30 days

• Support:
  o UML
  o SysML
  o ArchiMate
  o Entity Relationship Diagram (ERD)
  o Data Flow Diagram (DFD)

• Homepage:
IBM Rational Rhapsody

• Description:
  o A proven solution for modeling and design activities.

• Features:
  o Visual software development environment, Collaborative development, Model-based testing, Management and traceability for integrated requirements
  o 90 day trial or Academic license

• Support:
  o UML
  o SysML
  o AUTOSAR
  o DoDAF
  o MODAF

• Homepage:
48 practitioners

Use of ADLs:
- 86% use UML or an UML profile,
- 9% use ad hoc or in-house languages (e.g., AADL, ArchiMate)
- 5% do not use any ADL

Needs of ADLs:
- Design (~66%), communication support (~36%), and analysis support (~30%)
- Code generation and deployment support (~12%) and development process and methods support (~18%)

Limitations of ADLs:
- Insufficient expressiveness for non-functional properties (~37%)
- Insufficient communication support for non-architects (~25%)
- Lack of formality (~18%)

State-of-the-practice

What industry needs from architectural languages?

Formalizing software architecture descriptions
  o Models must be scalable
  o Multiple formal methods must be supported
    ▪ using multiple ADLs to model a single system
    ▪ formalizing different aspects of a system in a single ADL
  o Incremental formalization must be supported
    ▪ how do you formalize in the face of incompleteness?
    ▪ **Formalize only and exactly as much as necessary**
  o Analysis results must be transferable to design and implementation
    ▪ what good is deadlock detection at architecture alone?

Source: [http://sunset.usc.edu/classes/cs599_2000/September7b.pdf](http://sunset.usc.edu/classes/cs599_2000/September7b.pdf)
Future Directions
Future Directions

• Additional perspectives for describing software architectures
  o Runtime
  o Dynamic
  o Mobile

• Language features
  o Support multiple views
  o Customizations
  o Programming facilities

• Tools
  o Automated analysis
  o Architecture-centric development
  o Large-view management
  o Collaboration
  o Versioning
  o Knowledge management

Bibliography