Architecture Description Languages

Stefan Björnander
The Department of Computer Science and Electronics
Mälardalen University

Abstract

An *Architecture Description Language* (ADL) is a language designed to model a system. They have often graphical as well as plain text syntax. This paper starts with a brief description of ADLs, and then follows an overview of the popular ADLs on the market of today and conferences about ADLs. Finally, a connection to my own research is described.

Table of Contents

Introduction

Architecture Description Languages (ADLs) are computer languages describing the software and hardware architecture of a system. The description may cover software features such as processes, threads, data, and subprograms as well as hardware component such as processors, devices, buses, and memory. The connections between the components can be described in logical as well as physical terms.

The difference between an ADL and a programming language or a modelling language is not totally clear. However, according to Wikipedia (http://en.wikipedia.org/wiki/Main_Page, search word *Architecture Description Language*), there are some requirements for a language to be classified as an ADL:

- It should be suitable for communicating architecture to all interested parties.
- It should support the tasks of architecture creation, refinement and validation.
- It should provide a basis for further implementation, so it must be able to add information to the ADL specification to enable the final system specification to be derived from the ADL.
- It should provide the ability to represent most of the common architectural styles.
- It should support analytical capabilities or provide quick generating prototype implementations.

Most ADLs have some common features:

- They have a graphical syntax as well a formally defined syntax and semantics.
- They often have features for modeling distributed systems.
- Little support for capturing design information, except through general purpose annotation mechanisms
- They often have ability to represent hierarchical levels of detail including the creation of substructures by instantiating templates.

ADLs have several advantages as well as disadvantages. One advance is that they are designed to represent architectures in a formal way. Another advantage is that they often are designed to be readable to both human and machines. A disadvantage is that there is not yet an agreement of what the ADLs shall represent, especially when it comes to the behaviour of the system.

A system is constituted by components. These components can be defined on a high or low level. For instance, a GPS receiver can be modelled as a simple device that on request returns the current longitude, latitude, and altitude. On the other hand, it can be modelled as a complex component with every part explicitly described. The designer can start by defining a rough model at a high level and then incrementally refine the model.

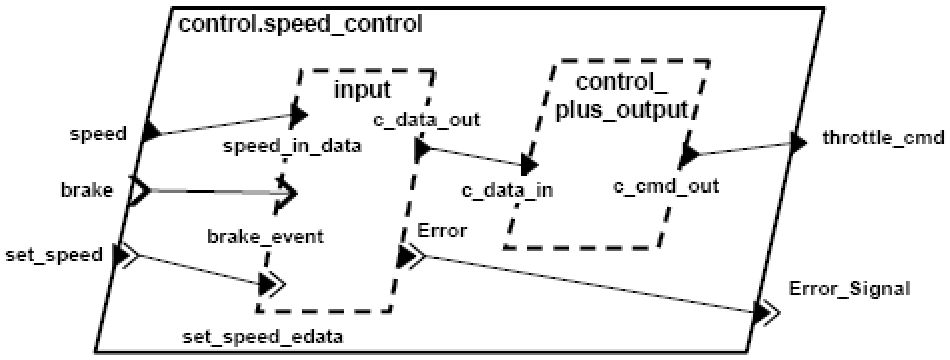
Most ADLs support aggregation; that is, one component can encapsulate other components. A common concept among ADLs is the division of a component into *interface* and *implementation*. The interface is the connection of the component to other component, such as ports. The implementation takes care of the intern parts of the component. The concept is similar to interfaces and classes in Java.

Languages

There are a lot of languages designed to address the problems of modelling a system. In this section, some of them are described.

AADL

Architecture Analysis & Design Language (AADL) is a large and complete language intended for design both the hardware and the software of a system. It supports processors, buses, devices, and ports as well as processes, threads, and data. It is possible to define physical port-to-port connections as well as logical flows through chains of ports. A process does not hold a thread by default; it has to be explicitly defined. Component definitions are divided into *component types* that define the features visible to other components and *component implementations* that define the inner parts of the component. Below is a graphical definition of a speed control system.



ACME

The Architectural Based Language and Environment (ACME) language is a small and rather simple language. It holds the concepts system, component, connector, port, role and representation. A system is constituted by components connected by connectors; the ports are end-points of the connectors. Actually, ACME can be considered a subset of AADL. One peculiar thing about ACME is that its representation can vary depending on the underlying model. For instance, a UNIX pipeline could be modelled in a syntax similar to C as follows:

```
Component pipe =
{
  Port in;
  Port out;
  Property implementation : String = “while (!in.eof)
                                      { in.read; compute; out.write; }”;
}
```

Rapide

Rapide is a set constituted by a type language, a definition language, a constraint language, and a executable programming language. The type language is intended to provide interfaces for the definition language, which defines the architecture. The constraint language defines requirements for timing and other pattern events. The executable language is concurrent and reactive. Its main purpose is to construct behaviour of components and connections between components.

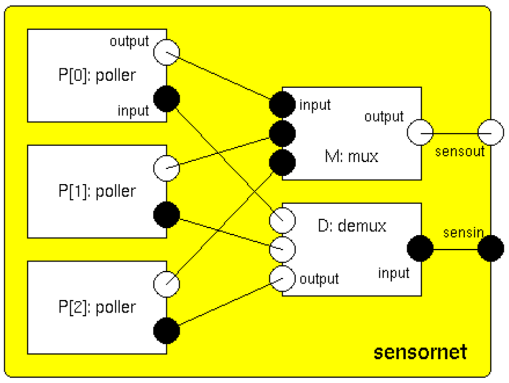
Wright

Wright is built upon the abstractions components, connectors, and configurations. The configurations can be divided into instances (a type if a specification of a component), attachments (describes the topology of the system), and hierarchy (a component may hold other components). Below is an example of a filter routine.

```
Configuration Capitalize
Component UpperCase
Connector Pipe
Instances
  Split : SplitFilter
  Upper : UpperCase
  Merge : MergeFilter
  P1, P2, P3 : Pipe
Attachments
  Split.Left as P1.Source
  Upper.Input as P1.Sink
  Split.Right as P2.Source
  Merge.Right as P2.Sink
  Upper.Output as P3.Source
  Merge.Left as P3.Sink
End Capitalize.
```

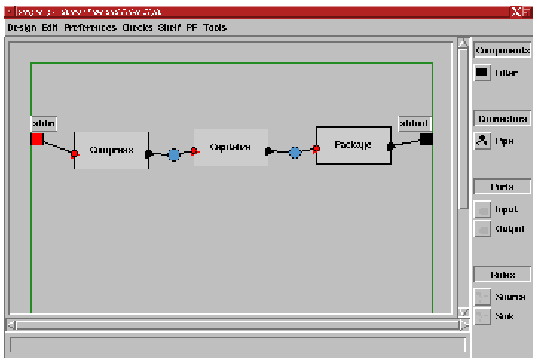
Darwin

Darwin supports *hierarchical composition.* One component is composed of other components or of *primitive* components; that is, built-in features of the language. Darwin also supports the structure of parallel programs and modelling of network topologies.



Aesop

The Software Architecture Design Environment Generator (Aesop) is a set of tools designed to develop a system model. It is based on the UNIX environment; it has pipe and filter style extensions in order to model those futures. However, it has a generic kernel, suitable for all environments. It has also a generic real-time extension. The tool does not provide a plain text description of the model; all modelling is done in the graphic editor of the tool. Below is an excerpt of a model.



UML

The Unified Modeling Language (UML) is a popular and widely spread modeling language. It was developed by James Rumbaugh, Grady Booch, and Ivar Jacobson (often referred to as *The Three Amigos* due to their infamous habit of arguing with each other) at Rational Software in the nineties. Technically, UML is not an ADL and was not inteded to be. However, it is rather suatible to model a system. Its main advantage is naturally that it is a widely spread language that many designer have knowledge in. Version 2.0 of UML was released in 2004. The release included the Object Constraint Language (OCL) that is a declarative language intended to describe the model in plain text. This release has, however, been criticized for being to large and complex to understand with a reasonable effort.

TASM

Timed Abstract State Machine (TASM) is strictly speaking not an ADL. However, it can be used to model a fairly complex system. As the name implies, it is based on the Abstract State Machine with the extension of timed features. A model of a system is constituted by a set of monitored variables that are read, a set of controlled variables that are written, and a set of rules. The machine (model) of the system starts at a given state, and the rules changes the state. The rules can be compared with an if-else-chain in a traditional language. The syntax of the language is inspired by Pascal and Ada. The variable *t* in the example below defines the time span of the operation of the rule, which can be a fixed number of logical time unites or an interval; it can also be the value *next*, which simple means that the operation waits for the next logical time step.

```
MAIN MACHINE:
  LIGHT_CONTROL

MONITORED VARIABLES:
  light_switch;

CONTROLLED VARIABLES:
  light;

RULES:
  R1: Turn On
      t := [4, 10];
      memory := 300;
      processor := 25;

      if light = OFF and light_switch = UP then
        light := ON;

  R2: Turn Off
      t := 6;
      memory := 100;
      processor := 15;

      if light = ON and light_switch = DOWN then
        light := OFF;

  R3: Else
      t := next;
      else then
        skip;
```

Conferences

There are several conferences more or less focused on ADLs.

ACM IEEE Design Automation Conference

The Design Automation Conference (DAC) is the foremost Electronic Design Automation (EDA) solution conference. On its last occasion, in June 2007, speeches were held over subjects revolving around design of automobile hardware. It also featured workshops for UML, Low Power Coalition, Design and Verification of Low Power ICs, and Hardware Dependent Software.

Working IEEE/IFIP Conference on Software Architecture

Working IEEE/IFIP Conference on Software Architecture (WICSA) is an international conference with focus on Software Architecture. On it last occasion, in January 2007, talks were held over Unified Process (UP) and a Management Perspective Software Architecture. Tutorials were also given about Pattern Oriented Software Architecture, Evaluating Software Architectures, Performance Analysis of Distributed Software Systems, and Refactoring Methods.

The IEEE Real-Time and Embedded Technology and Applications Symposium

In 2004, the IEEE Real-Time and Embedded Technology and Applications Symposium gave a workshop over Model-Driven Embedded Systems (MoDES '04). The workshop address three issues: MoDES challenges in industrial practice, Recent research advances in MoDES, and Crossing the chasm between MoDES research and practice.

World Computer Congress

When World Computer Congress (WCC) was held in 2004, there was a workshop given on the theme Architecture Description Languages. It continued discussions on Models and Analyses, Specification and Design, and Domain Specific ADLs.

Relevance for my own Research

In consultation with my supervisor Kristina Lundqvist, I plan to develop a translation tool between AADL and TASM. The languages are quite different in organisation and features. However, they can both be used for modelling a system.

References

Architecture Description Languages from Wikipedia, the free encyclopaedia

http://en.wikipedia.org/wiki/Architecture_Description_Languages

The following papers were presented at the workshop Model-Driven Embedded Systems (MoDES '04) at the IEEE Real-Time and Embedded Technology and Applications Symposium.

An overview of the SAE Architecture Analysis Design Language (AADL) Standard: a basis for model-based architecture-driven embedded systems engineering - Peter H. Feiler (Software Engineering Institute, USA), Bruce Lewis (US Army, USA), Steve Vestal (Honeywell Laboratories Minneapolis, USA), Ed Colbert (Absolute Software, USA)

Deploying QoS Contracts in the Architectural Level - Sidney Ansaloni (U. Fluminense, Brazil), Alexandre Sztajnberg (U. Rio de Janeiro, Brazil), Romulo Curty (U. Fluminense, Brazil), Orlando Loques (U. Fluminense, Brazil)

Hierarchical Composition and Abstraction in Architecture Models - Pam Binns and Steve Vestal (Honeywell Laboratories Minneapolis, USA)

Pattern-Based Analysis of an Embedded Real-time System Architecture - Peter H. Feiler (Software Engineering Institute, USA), David P. Gluch (Embry-Riddle U., USA), John J. Hudak (Software Engineering Institute, USA), Bruce A. Lewis (US Army, USA)

An ADL centric approach for the formal design of real-time systems - Sébastien Faucou , Anne-Marie Déplanche, Yvon Trinquet (RTSt -IRCCyN, France)

SafArchie Studio: ArgoUML Extensions to Build Safe Architectures - Olivier Barais and Laurence Duchien (LIFL U. Lille, France)

Enhancing the Role of Interfaces in Software Architecture Description Languages (ADLs) - Seamus Galvin, Chris Exton, Finbar McGurren (U. Limerick, Ireland)

How ADLs can help in adapting the CORBA Component Model to Real-Time Embedded Software Design - Sylvain Robert (CEA/SACLAY, France), Vincent Seignole(Thales, France), Sébastien Gérard (CEA/SACLAY, France), Stéphane Ménoret, Virginie Watine(Thales, France), Ansgar Radermacher, François Terrier (CEA/SACLAY, France)

Specification of Intel IA-32 using an Architecture Description Language - Jeff Bastian, Soner Onder (Texas Instruments, USA) *Cotre as an AADL profile* - Patrick Farail, Pierre Gaufillet (Airbus, France)

EAST-ADL an Architecture Description Language, Validation and Verification Aspects - EAST-ADL an Architecture Description Language, Validation and Verification Aspects - Vincent Debruyne (PSA Peugeot-Citroën, France) Françoise Simonot-Lion (LORIA, France), Yvon Trinquet (IRCCyN, France)