Architecture description language

From Wikipedia, the free encyclopedia

Different communities use the term architecture description language. Two important communities are:

- The software engineering community
- The enterprise modelling and engineering community

In the software engineering community, an **Architecture Description Language** (**ADL**) is a computer language used to describe software and/or system architectures. This means in case of technical architecture, the architecture must be communicated to software developers. With functional architecture, the software architecture is communicated with stakeholders and enterprise engineers. By the software engineering community several ADLs have been developed, such as Acme (developed by CMU), AADL (standardized by SAE), C2 (developed by UCI), Darwin (developed by Imperial College London), and Wright (developed by CMU).

The enterprise modelling and engineering community have also developed architecture description languages catered for at the enterprise level. Examples include ArchiMate (now an Open Group standard), DEMO, ABACUS (developed by the University of Technology, Sydney) etc. These languages do not necessarily refer to software components, etc. Most of them, however, refer to an application architecture as the architecture that is communicated to the software engineers.

Most of the writing below refers primarily to the perspective from the software engineering community.

Contents

Introduction

The following reasons would be better served if there were a standard notation (ADL) for representing architectures:

- Mutual communication
- Embodiment of early design decisions
- Transferable abstraction of a system

Architectures in the past were largely represented by box-and-line drawing. The following things are usually defined in such a drawing:

- Nature of the component
- Component properties
- Semantics of connections
- Behavior of a system as a whole

ADLs result from a linguistic approach to the formal representation of architectures, and as such they address its shortcomings. Also important, sophisticated ADLs allow for early analysis and feasibility testing of architectural design decisions.

Characteristics

There is a large variety in ADLs developed by either academic or industrial groups. Many languages were not intended to be an ADL, but they turn out to be suitable for representing and analyzing an architecture. In principle ADLs differ from:

- Requirements languages, because ADLs are rooted in the solution space, whereas requirements describe problem spaces.
- Programming languages, because ADLs do not bind architectural abstractions to specific point solutions
- Modeling languages, because ADLs tend to focus on representation of components rather than the behavior of the whole. However, there are domain specific modeling languages (DSMLs) that focus on representation of components.

The following list is a minimal set of requirements for a language to be an ADL. The languages must :

- Be suitable for communicating an architecture to all interested parties
- Support the tasks of architecture creation, refinement and validation
- 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
- Provide the ability to represent most of the common architectural styles
- Support analytical capabilities or provide quick generating prototype implementations

ADLs have in common:

- Graphical syntax with often a textual form and a formally defined syntax and semantics
- Features for modeling distributed systems
- Little support for capturing design information, except through general purpose annotation mechanisms
- Ability to represent hierarchical levels of detail including the creation of substructures by instantiating templates

ADLs differ in their ability to:

- Handle real-time constructs, such as deadlines and task priorities, at the architectural level
- Support the specification of different architectural styles. Few handle object oriented class inheritance or dynamic architectures
- Support analysis
- Handle different instantiations of the same architecture, in relation to product line architectures

Positive elements of ADL

- ADLs represent a formal way of representing architecture
- ADLs are intended to be both human and machine readable
- ADLs support describing a system at a higher level than previously possible
- ADLs permit analysis of architectures – completeness, consistency, ambiguity, and performance
- ADLs can support automatic generation of software systems

Negative elements of ADL

- There is not universal agreement on what ADLs should represent, particularly as regards the behavior of the architecture
- Representations currently in use are relatively difficult to parse and are not supported by commercial tools
- Most ADLs tend to be very vertically optimized toward a particular kind of analysis

Common concepts of architecture

The ADL community generally agrees that Software Architecture is a set of components and the connections among them. But there are different kind of architectures like :

Object Connection Architecture

- Configuration consists of the interfaces and connections of an object-oriented system
- Interfaces specify the features that must be provided by modules conforming to an interface
- Connections represented by interfaces together with call graph
- Conformance usually enforced by the programming language
  - Decomposition - associating interfaces with unique modules
  - Interface conformance - static checking of syntactic rules
  - Communication integrity - visibility between modules

Interface Connection Architecture

- Expands the role of interfaces and connections
  - Interfaces specify both “required” and “provided” features
  - Connections are defined between “required” features and “provided” features
- Consists of interfaces, connections and constraints
  - Constraints restrict behavior of interfaces and connections in an architecture
  - Constraints in an architecture map to requirements for a system

Most ADLs implement an interface connection architecture.

Architecture vs. design

So what is the difference between architecture and design? Architecture casts non-functional decisions and partitions functional requirements, whereas design is a principle through which functional requirements are accomplished. The process of defining an architecture may use heuristics or iterative improvements; this may require going a level deeper to validate the choices, so the architect often has to do a high-level design to validate the partitioning.

Examples

Below the list gives the candidates for being the best ADL until now

- Primary candidates
  - ACME / ADML (CMU/USC) (http://www-2.cs.cmu.edu/~acme/)
  - Rapide (Stanford) (http://complexevents.com/stanford/rapide/)
  - Wright (CMU) (http://www-2.cs.cmu.edu/afs/cs/project/able/www/wright/index.html)
  - Unicon (CMU)
  - LePUS3 and Class-Z (University of Essex) (http://lepus.org.uk/)
  - ABACUS (UTS) (http://www.avolution.com.au)
- Secondary candidates
  - Aesop (CMU) (http://www.cs.cmu.edu/afs/cs/project/able/www/aesop/aesop_home.html)
  - MetaH (Honeywell)
  - AADL (SAE) - **A**rchitecture **A**nalysis & **D**esign **L**anguage
  - C2 SADL (UCI)
  - SADL (SRI) - **S**ystem **A**rchitecture **D**escription **L**anguage
- Others
  - Lileanna - **L**ibrary **I**nterconnect **L**anguage **E**xtended with **Ann**otated **A**da
  - UML - **U**nified **M**odeling **L**anguage

Approaches to architecture

Approaches to Architecture

- Academic Approach
  - focus on analytic evaluation of architectural models
  - individual models
  - rigorous modeling notations
  - powerful analysis techniques
  - depth over breadth
  - special-purpose solutions
- Industrial Approach
  - focus on wide range of development issues
  - families of models
  - practicality over rigor
  - architecture as the big picture in development
  - breadth over depth
  - general-purpose solutions

Conclusion

- There is a rich body of research to draw upon
- Much has been learned about representing and analyzing architectures
- Effort is needed now to bring together the common knowledge and put it into practice

See also

- AADL
- Darwin
- ABACUS
- Scripting language

References and External links

- ArchiMate (http://www.archimate.org) An example of an ADL for enterprise architecture
- DEMO (http://www.amazon.com/dp/3540291695) Another example of an enterprise architecture ADL
- A Classification and Comparison Framework for Software Architecture Description Languages (http://citeseer.ist.psu.edu/163640.html) - 1997 research paper.
- ACME (http://www.cs.cmu.edu/~acme)
- ABACUS (http://www.avolution.com.au)
- Rapide (http://complexevents.com/stanford/rapide/)
- Wright (http://www.cs.cmu.edu/afs/cs/project/able/www/wright/index.html)
- Aesop (http://www.cs.cmu.edu/afs/cs/project/able/www/aesop/aesop_home.html)
- Unicon (http://www.cs.cmu.edu/afs/cs/project/vit/www/unicon/index.html)
- C2 SADL (http://www.ics.uci.edu/pub/arch/)
- SSEP (http://www.mcc.com/projects/ssepp)
- ADML (http://www.mcc.com/projects/ssepp/adml)
- DAOP-ADL (http://caosd.lcc.uma.es/CAM-DAOP/DAOP-ADL.htm)
- AO-ADL (http://caosd.lcc.uma.es/AO-ADL.htm)
- DiaSpec (http://diaspec.bordeaux.inria.fr/) an approach and tool to generate a distributed framework from a software architecture

Retrieved from "http://en.wikipedia.org/wiki/Architecture_description_language"
Categories: Software architecture