COMPARISON OF FIVE ARCHITECTURE DESCRIPTION LANGUAGES ON DESIGN FOCUS, SECURITY AND STYLE

Csaba Egyhazy
Computer Science Department, Virginia Tech, 7054 Haycock Road, Falls Church, Virginia 22043, USA

Keywords: Architecture Description Language, high-level abstraction, component, connector, configuration, interface.

Abstract: With the increasing complexity and size of software systems, defining and specifying software architectures becomes an important part of the software development process. In the past, many software architectures have been described and modeled in an ad hoc and informal manner. For the past 20 years, Architecture Description Languages (ADLs) have been proposed to facilitate the description and modeling of software architectures. This paper reviews the history of ADLs, selects five of them, and compares them based on their design focus, security modeling, and styles modelling.

1 INTRODUCTION

Architecture Description Languages (ADLs) are formal notations which are used to describe and model software architectures (Shaw and Garlan, 1996), (Perry and Wolf, 1992). However, ADLs are actually far more than simple language syntax; *"... this notations usually provide both a conceptual framework and a concrete syntax for characterizing software architectures. They also typically provide tools for parsing, unparsing, displaying, compiling, analyzing, or simulating architectural descriptions written in their associate language ... "* (Garlan and Perry, 1995).

Shaw and Garlan describe an ADL as a language which has precise descriptions and specifications: *"provides models, notations, and tools to describe architectural components and their interactions; it must handle large-scale, high level designs; it must support the adaptation of these designs to specific implementations; it must support user-defined or application specific abstractions; and it must support principled selection of architectural paradigms... "* (Shaw and Garlan, 1992).

The first comprehensive survey of ADLs was done by Clement in 1996 (Clements, 1996). About the same time, Nenad Medividovic and Richard N. Taylor published a framework **Error! Reference source not found.** for classifying and comparing ADLs. Since then, some of these ADLs have been developed to new generation (e.g. ACME, MetaH) and some of them declined (e.g. UniCon, Wright, C2). New ADLs such as ArchWare and xADL 2.0 emerged. Consequently, there is a need for a comparative review of some of these contemporary ADLs at this time. In this paper we provide such a comparison with respect to three metrics: design focus, security modeling, and style modeling.

2 HISTORICAL VIEW OF ADLS

Our review of the evolution of ADLs resulted in a taxonomy marked by three periods: theory base period, first-generation ADLs, and second-generation ADLs. As seen from Figure 1 starting at the top, the Theory Base Period commenced in the late 1980s with the publication of Mary Shaw's paper on "high-level abstraction" and software architecture style concepts 0). Since then, research expands on architecture level of abstractions including the decomposition of architectures, the interconnection between components, the notation for describing architectures, and the formal methods adopted in describing components, behaviors, and connections. We call this period the Theory Base Period. Two methodologies of modeling the software architecture form the conceptual basis of Theory Base Period. The one founded by Mary Shaw and David Garland, and the other by Perry and Wolf (Perry and Wolf, 1992). The Shaw and Garlan (Shaw and Garlan, 1996) methodology is built

Egyhazy C. (2007).
COMPARISON OF FIVE ARCHITECTURE DESCRIPTION LANGUAGES ON DESIGN FOCUS, SECURITY AND STYLE.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - ISAS*, pages 270-277
DOI: 10.5220/0002348002700277

around a series of styles. The idea of styles comes from the concept of pattern in high-level programming languages. The Perry and Wolf (Perry and Wolf, 1992) methodology is built around the idea that software systems can be represented as a triple {Element, Rule, Rationale}. During the first part of the 1990s software architecture formalization focused on individual systems or styles. Examples include the formalization of pipe-filter style (Luckham, Vera, and Meldal, 1995), event systems **Error! Reference source not found.**) and object-composition standard (Garlan and Notkin, 2001). Therefore, the cost of development using such formalization actually increases. Second, there are no corresponding tools to support the formalization and because they develop the semantics of a style from scratch, there is no direct means of characterizing a specific system configuration.

First-generation ADLs focus more on the modeling of architecture elements. The modeling is based on the gross structure proposed by Shaw and Garland, where an ADL describes the architecture in terms of component, connector and the way they are composed with each other. Components can be described with specification and implementation. So do connectors. Components are the loci of computation and state. Each component has an interface specified with component type and the player. Player is called an interface point in this paper. Connectors are the loci of relations among components. They mediate interactions. Each connector has a protocol specification which is the interface of the connector provided to components or other connectors. The protocol includes rules about the types of interface, assurances about properties of the interaction, rules about the order in which things happen, and commitments about the interaction.

As seen in Figure 1, ADLs such as UniCon, Wright, ACME, Rapide, SRI SADL, C2 ADL and MetaH belong to the first-generation ADLs. UniCon was developed around 1995 by a CMU group lead by Mary Shaw. It is in UniCon where the aforementioned gross structure of architecture language was proposed. UniCon is a consequence of Mary Shaw's research on architecture styles. UniCon, implements the pipe and filters architecture style, real-time scheduling architecture, and global data access architecture. Wright focused on the concept of explicit connector types, on the use of automated checking of architectural properties.

Rapide's original goal was to build large-scale distributed multi-language systems. Rapide adopts an event-based execution model of distributed time-sensitive systems. This model is called the "timed poset model." Posets provide the most detailed formal basis to date for constructing early life cycle prototyping tools, and later life cycle tools for correctness and performance analysis of distributed time-sensitive systems.

Compared with other first-generation ADLs, Structural Architecture Description Language (SADL) is relatively new. Its research mainly focuses on the architecture styles and architecture refinement patterns. The focus of MetaH (Honeywell MetaH Website) is to build reliable, real-time multiprocessor avionics system architectures. ACME was proposed to provide a structural framework for characterizing architectures, together with annotation facilities for additional ADL-specific information.

Most of the second-generation ADLs derived from a specific first-generation ADL. For example, on the base of MetaH, Honeywell Inc. developed AADL; CMU and University of California at Irvine united and created xADL 2.0 based on ACME and C2 SADL research; ADML is an XML improved version of ACME. Most of second generation ADLs have a relatively complete tools support (Bass, Clements, Kazman, 2003). They are more extensible, such as xADL 2.0, and focus on handling dynamic architectures, e.g., ArchWare.

3 FIVE ARCHITECTURE DESCRIPTION LANGUAGES

In our comparative review of ADLs, we selected two ADLs from the first-generation and three ADLS from the second generation for this comparison. They are: Rapide, SADL, xADL 2.0, AADL and ArchWare. Among the second-generation ADLs, we selected xADL 2.0, derived from xArch. Since AADL inherited most of the characteristics of MetaH, there is no need to include ACME, C2 SADL and MetaH in our comparison. Also, from the aforementioned history we know that UniCon, Wright, Aesop and ACME were all generated by the CMU research group, and are therefore similar to each other. Most of their characteristics are inherited by the second generation ADLs such as xADL 2.0 and AADL. On the other hand, our other selections, namely Rapide and SADL were developed independently and are different from the ADLs developed at CMU.

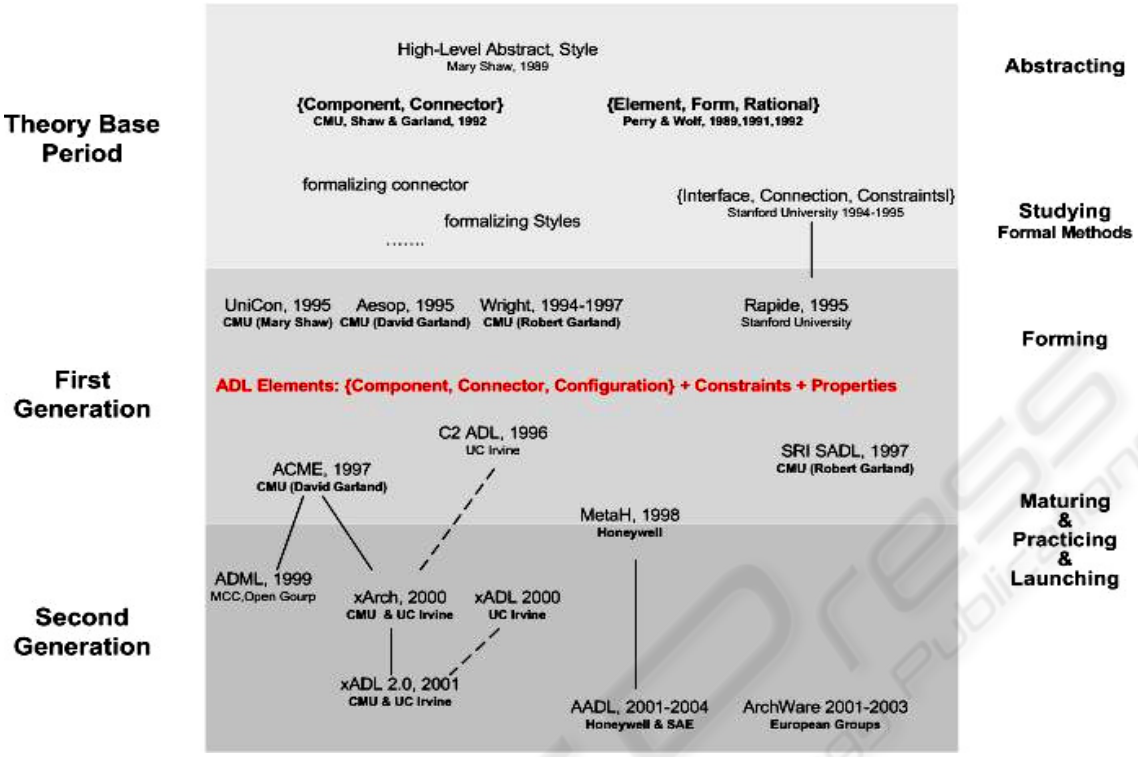

Figure 1: Architecture Description Language Evolution Chronicle.

Our last selection was ArchWare, the newest ADL in our survey. It focuses on modeling active architectures, and it has many features which are not shown in other ADLs.

3.1 Rapide

Rapide abstracts architecture as an interface connection architecture 0 (Dashofy E. M., van der Hoek A. and Taylor R. N., 2002). According to this abstraction, an architecture consists of interface, connections and constraints. Interfaces are the feature for component abstraction. Connections define the communication between components, and Constraints restrict the behavior of the interfaces and connections. Rapide is an event processing language, so the semantics of Rapide are defined in terms of generating events, sending events from one component to another component and observing events. A component in Rapide is described in terms of interface. It includes the kinds of events that a component can observe or generate, the functions the component provides to other components or requires from other components, and the states and state transitions of the component and constraints on its external behavior. The interface declares sets of constituents to represent behaviors contained in the components which are visible to the external architecture.

3.2 SADL

SADL adopted the common way of architecture abstraction (Moriconi and Riemenschneider, 1997), namely one represented with components, connectors and configurations. The component has a name, a type and an interface. The configuration defines the way of wiring components and connectors into an architecture. A configuration contains two kinds of elements: connections and constraints. Mapping is very important in SADL. It is a relation that defines a syntactical interpretation from the abstract level of architecture to the concrete level of architecture.

In SRI SADL, a component is a sub-type of type component. A component has a name, a type and an interface. The interface points of a component are represented with ports. A port has a name and a type, and is designated for input or output. Input port is tagged as iport. Output port is tagged as oport.

SADL supports sub-typing as a means of defining new architectural objects within a particular class of objects. For example, component is the

built-in type of SADL. SADL component type supports type constraints. In addition, SADL is algebraically extensible. It is possible to introduce new types that are not derived from a predefined collection of base types. Semantics in SADL are represented over constraints and properties. Constraints in SADL are first-class objects using classical first-order logic with quantifiers that range over the natural numbers. The SADL Constraint Language provides a means to describe formal semantics.

3.3 xADL 2.0

xADL 2.0 separates an architecture description into the design time description and the runtime description. Architectures are modeled with a set of schemas. These schemas have the following dependencies. Instance Schema is used to represent run-time elements. Design time elements are represented in Structure and Types Schema. The Options Schema provides the ability to specify that certain components, connectors and links are optional when instantiating architecture. The Variants Schema provides the ability to specify that the type of certain components and connectors can vary when instantiating the architecture. The Version Schema provides the ability to store version evolution information of elements in an architecture. In xADL 2.0, an architecture is represented with components, connectors and links (Dashofy, E. M, 2003).

In xADL 2.0, component, connector, interface and link instance are semantically neutral, which means that their behaviors are not formally specified. Behavior specification can be specified in extensions. As well as the semantics, xADL 2.0 does not take any steps to define constraints or rules about components and their behaviors. Constraints can be specified in extensions.

3.4 AADL

AADL was explicitly designed to meet the needs of embedded real-time systems developed by model-based engineering, such as avionics, automotive electronics and robotic systems. These systems are usually performance critical.

An AADL specification consists of a global declaration and an AADL declaration. The global declaration includes package specifications and property set declarations. AADL declarations include component type, component implementation, port group types, and annex libraries. A component type specifies a functional interface in terms of features flow specifications and properties. The feature describes the interface of a component through which control and data may be exchanged with other components. As shown in the Figure 2, data, subprograms, threads, thread groups and processes collectively represent the application software. They are called software components. Processor, memory, bus and device collectively represent the execution platform. They are called execution platform components.

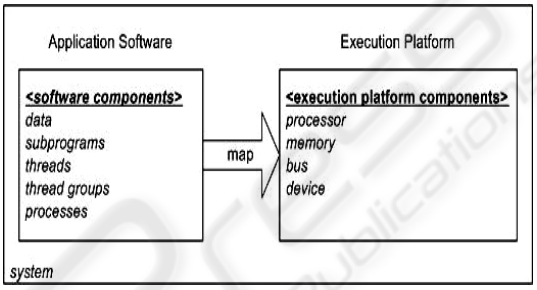

Figure 2: Components from Application Software to Execution Platform.

3.5 ArchWare

In ArchWare, an architecture is described in terms of components, connectors and their compositions (ArchWare Project Site). Components are described in terms of external ports and an internal behavior. Connectors are special-purpose components. This abstraction is represented pictorially in Figure 3.

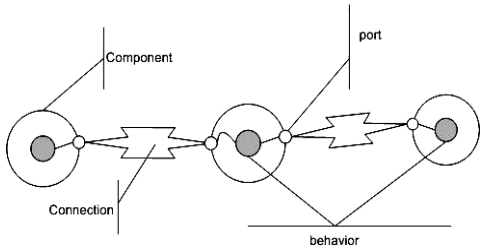

Figure 3: Architectural Concept of ArchWare (Adopted from ArchWare Project Site).

The hyper-code abstraction was introduced in ArchWare as a mean of unifying the concept of source code, executable code and data in a programming system (Ron, et.al, 2005). Since it can present closure, through sharing links, it can be used as a representation for introspection of the executing system. As seen from Figure 4, ArchWare is a formal type system consisted of three layers. With this type mechanism, it is easy to extend ArchWare

with new base types and new type constructors. Therefore, ArchWare can be seen as an open family of layered languages having as root the base layer. The type of a component is represented by the port type and behavior type. Both types can be parameterized. A port type defines a communication protocol.

4 COMPARISON BASED ON DESIGN FOCUS, SECURITY AND STYLE

In the study reported here, we used a comparison framework that consisted of several metrics grouped according to 3 general features: Design Focus, Security, and Style.

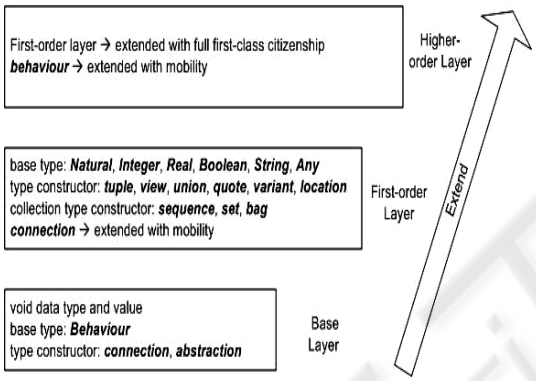

Figure 4: Three Layers of Type Definition.

A comparison outline table is given in Figure 5. Each ADL has its own design goal. Basically, there is no ADL that can be adapted to every situation.

4.1 Design Focus

ADLs give us an insight into which area they are more competent in. It is our hope that the comparison given in Figure 5 will facilitate the selection of an ADL over another when satisfying particular development project requirements.
Rapide focuses on large-scale distributed and time-sensitive systems. Rapide demonstrated to be used in modeling NSA's Multilevel Systems Security Initiative, X/Open Distributed Transaction Processing Industry Standard. SRI SADL was designed for general purpose. Most of ADLs except AADL we surveyed can be used in general domains, but they have different design goals. xADL focuses on extensibility and flexibility architectures. Rapide focuses on large-scale distributed, time-sensitive, and multi-language systems, particularly in solving distributed system design problems. So even though it can be used in designing general systems, it is more applicable in distributed multi-languages. SRI SADL specializes in refinement. For mobile software system, ArchWare is superior. AADL is specific to embedded system. xADL 2.0 can be used as a bridge between different ADLs.

ADL Metrics	Rapide	SADL	xADL	AADL	ArchWare
Design Focus	Large-scale distributed, time-sensitive, multi-language systems	No particular domain requirement, focus on refinement	Focus on extensibility and flexibility architectures	Focus on embedded real-time systems, high reliability, timing, responsiveness, throughput, safety requirements	Focus on active software architecture
Security Modeling	Not provided	Theoretically proved that when security plans are set up in low level abstraction, it will be effective in higher level too	Use the extensible foundation of the language; support architectural access control module	Can define safety engineering concepts (MTBF, propagation through connections) Fail secure attribute model etc.	Not provided
Style Modeling	Defined with the architecture section. Support parameterization. Events can be styled as event pattern	Defined with the ARCHITECTURE section. Use predicate mapping to map styles.	Defined with a XML schema No semantic supported	Defined with the system section. The system has multiple modes with each representing a possible configuration case	A style consists of types, constituent elements, constraints. It is a property-guarded abstraction

Figure 5: Comparison of Five ADLs on General Features.

4.2 Security Modeling

In traditional software designs, security is often considered as an extension or a remedy rather than a part of the architecture, it is relatively acceptable when the system is simple and homogeneous. However, with the system becoming larger and larger, more and more software is built from existing components. These components may come from different sources. Consequently, the practice of component-based software engineering requires architects to consider security not only as a property of individual components, but also as an important issue in the architecture as a whole (Ren and Taylor, 2005).

Security issues are seldom included in the consideration from the language point of view. Integrating security into the architecture modeling introduces a degree of complexity. However, it can guide the comprehensive development of security. Such high-level modeling enables designers to locate potential vulnerabilities and install appropriate countermeasures. It facilitates checking that security is not compromised by individual components and enables secure interaction between components. It also enables the possibility of selecting the most secure components and supports continuous refinement. In recent years, security modeling has received greater attention, particularly in the development of second-generation ADLs. If an ADL does not provide explicit security modeling, it does not mean that the ADL is incapable of handling security issues. It simply means that it was not originally conceived to address security in particular. For example, the SADL group proved that if security properties were provided at the lower level in the architecture's hierarchy, they could be persevered at higher levels by using formalizing mappings and correctness arguments. They illustrated this idea by applying SADL on enforcing the multilevel security (MLS) policy in the X/Open Distributed Transaction Processing architecture. However, SADL only proved that they are capable of enacting security policies. They didn't introduce any security policies or specify any security properties in the language itself.

xADL 2.0 modeled two types of access privileges (Dashofy, E. M, 2003). The first type handles passive resources. It deals with traditional access such as read and write. The second type handles active resources. It includes control on instantiation and destruction of architectural constituents, connection of components with connectors, execution and reading and writing of architecturally critical information. Traditional ADLs have paid little attention to the second type. However, they are obviously important from the architectural point of view. Safeguard refers to permissions that are required to access the interfaces of the protected components and connectors. A safeguard specifies what prerequisite other components or connectors should have before they access a certain protected component or connector. Policy specifies what privileges a subject should have to access resources protected by safeguard. It can be regarded as a specific security solution for a component or a connector.

Components and connectors play different roles in the security scheme. Components work as the supplier of the security contract. Connectors play an important role in regulating and enforcing the security contract specified by components. It can decide what subjects the connected components are executing for, and regulate whether components have sufficient privileges to communicate through the connectors. It also has the potential to provide secure interaction between insecure components.

For example, AADL's optional set of declarations and semantics can be used to introduce new properties of components that support the addition of security techniques. The error model annex in AADL is able to define qualitative and quantitative analysis of non-functional requirements, such as security requirement and safety requirement on both components and connectors. To fully support architectural security design, the error annex should provide security modeling tools that can be integrated with the AADL tools. This has not been realized yet (Feiler, Gluch, Hudak, and Lewis, 2004).

4.3 Style Modeling

An architecture style captures common computation and communication paradigms used to address a particular class of programming problems. A mature and formal architecture style should possess three basic elements: a) A well-defined notation for capturing architectures developed in the style b) Well-defined methods for producing and analyzing formal models from a specification captured in the notation, and c) A well-defined method for producing an implementation from a specification capture in the notation.

A style should contain precise semantics; provide interface coding guidelines for source modules and provide well-defined methods for assembling components to produce an overall implementation.

This allows for the styles to be evaluated, verified and reused in different environments. The support for styles in an ADL is essential, and will affect its usability and scalability. Style is often equal to pattern in many cases.

Architecture style in Rapide is a set of interfaces (components), a set of connection rules and a set of constraints. Connection rules define relationships between events independently of any implementation; connections are defined using event patterns. The event patterns provide the expressive means to define both static and dynamic architecture. As seen in Figure 6, a style is directly expressed with architecture in Rapide.

```
with comp1, comp2
architecture styleA is
        ?C1: comp1; ?M : Msg;
        ?C2: comp2;

connections
```

Figure 6: A Style in Rapide.

As shown in Figure 7, style is defined with architecture in SADL. The architecture basically contains a components section, a connectors section and a configuration section. The architecture can import declarations of types, variables, constants, assertions and architectures from other specifications with import. The architecture can also export aforementioned elements as well. The architecture can be parameterized and can be added as a new type in SADL. SADL supports style mapping with predicates.

In xADL 2.0, the architecture type is composed of three collections: components, connectors and links. However, because xADL2.0 itself does not contain any semantic description, the architecture is regarded as a container of styles. At this time, the definition of an architecture style with complete meaning needs extensions (Dashofy, E. M, 2003).

```
styleA: ARCHITECTURE
        [char_iport : SEQ (character) → code_oport
: code]
        IMPORTING character, … , FROM type_1
        IMPORTING Function FROM
Function_Style
    ….
BEGIN
COMPONENTS
        comp1: …
        comp2: …
        ….
```

Figure 7: SADL style – ARCHITECTURE.

5 CONCLUSIONS

In this paper, we conducted a study of five ADLs based on design focus, styles, and security. Although security issues are gaining more and more attention in the ADL literature, few provide complete solutions. SADL and AADL claim that the languages are extensible so they are capable of defining security properties. However, without clear and pre-defined terminologies, it is hard for application architects to implement the extension. xADL 2.0 is the only one in our comparison that provides a security framework. However, we think there is an inherent shortcoming in xADL 2.0. For example, xADL 2.0 does not itself provide any semantic and constraints in the language, it makes their security control stop in the access level. Also, it suffers from the security issues brought by plug-in components or connectors described in other ADLs, especially when those components or connectors are composite. The security solution provided by xADL 2.0 is limited to access control. ADLs have begun to address security issues. However, it is still far from established.

Although all ADLs support the definition of architecture styles, the support levels are different. Most of them satisfied the first criteria, that is, they all have a well-defined notation for capturing architecture developed in style. The only exception is xADL 2.0. Because its design goal, it does not capture the semantic of an architecture. Rapide and SADL allow parameterization in defining styles. This may increase the reusability of a style. For example, in a Client/Server architecture style, the network protocol between the client and the server is parameterized. It can be TCP/IP or SNA by using parameter input. The parameter can be a number too, for example, the number of the client.

In closing, we believe that once one of the above ADLs emerges as the front runner, and becomes the likely de facto industry standard, an increase in the adoption of ADLs as part of the software engineering process will follow.

REFERENCES

Abowd G., Allen R. and Garlan D., 1993, Using Style to Understand Descriptions of Software Architecture. *Proceedings of the 1st ACM SIGSOFT symposium on Foundations of software engineering Pages: 9 – 20*

ArchWare Project Site: http://www.arch-ware.org/

Bass L., Clements P., and Kazman R., 2003, *Software Architecture in Practice*, Second Edition, Addison-Wesley Professional; 2 edition, April, 2003

Clements P. C., 1996, A Survey of Architecture Description Languages, *International Workshop on Software Specifications & Design, Proceedings of the 8th International Workshop on Software Specification and Design*

Dashofy E. M., van der Hoek A. and Taylor R. N., 2002, An Infrastructure for the Rapid Development of XML-based Architecture Description Languages*, In Proceedings of the 24th International Conference on Software Engineering*

Dashofy, E. M, 2003, *A Guide for Users of the xADL 2.0 Language First Revision*, Institute for Software Research at the University of California

Feiler P., Gluch D., Hudak J., Lewis B., 2004, *Embedded System Architecture Analysis Using SAE AADL*, Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst

Garlan D. and Notkin D., 2001, Formalizing design spaces: Implicit invocation mechanisms. In *VDM'91: Formal Software Development Methods, page 31-44*

Garlan D. and Perry D., 1995, Introduction to the special issue on *Software Architecture IEEE Transactions on Software Eng., April 1995.*

Garlan, D. Monroe R. and Wile D., 1997, ACME: An Architecture Description Language*, Proceedings of CASCON 97, Toronto, Ontario, November 1997, pp. 169-183.*

Honeywell MetaH Website: http://www.htc.honeywell.com/metah/

Luckham, D. C. Vera J., Meldal S., 1995, Three Concepts of System Architecture, *Technical Report: CSL-TR-95-674, 1995*

Medividovic N. and Taylor R. N., 1996, A Classification and Comparison Framework for Software Architecture Description Languages *IEEE Transactions on Software Engineering Volume 26, Issue 1, 1996*

Moriconi M. and Riemenschneider R. A., 1997, Introduction to SADL 1.0: A Language for Specifying Software Architecture Hierarchies, SRI International 1997

Perry D. E. and Wolf A. L., 1992, *Foundations for the Study of Software Architecture*, ACM SIGSOFT Software Engineering Notes, 1992 Publishing Company, Singapore, pp. 1-39, 1993

Rapide Design Team, 1997, *Draft: Rapide 1.0 Types*

Language Reference Manual, Program Analysis and Verification Group, Computer Science Lab, Stanford University

Ren, J. and Taylor R. N., 2005, *A Secure Software Architecture Description Language*, Department of Informatics, University of California, Irvine*,*

Shaw M. and Garlan D., 1996*, Software Architecture: Perspectives on an Emerging Discipline,* Prentice Hall, 1996, Page 167.

Shaw M. and Garlan D., 1993*, An Introduction to Software Architecture, Advances in Software Engineering and Knowledge Engineering,* Series on Software Engineering and Knowledge Engineering, Vol 2, World Scientific

Shaw M., 1989*, Larger Scale Systems Require Higher-Level Abstractions*, ACM, 1989

Ron Morrison, Graham Kirby, Dharini Balasubramaniam, Kath Mickan, Flavio Oquendo, Sorana Cimpan, Brian Warboys, Bob Snowdon and R Mark Greenwood, 2004, Support for Evolving Software Architectures in the ArchWare ADL, *IEEE/IFIP Conference on Software Architecture (WICSA).*