Architecture Description Languages and Information Systems Architects: Never the Twain Shall Meet?

IFIP WG 2.10 Software Architecture Meeting
February 21 - 25 2005, Vancouver, Canada

Eoin Woods
Zühlke Engineering Limited
ewo@zuhlke.com

Abstract

This short position paper briefly reviews the state of practice in architectural description for information systems, and asks why purpose designed architecture description languages are not more widely used in this domain. It then attempts to answer the question, from the author's perspective, by reviewing the needs that an information systems architect would have for a purpose designed architecture description language.

Introduction

In a quest to improve the practice of software architecture, many researchers have proposed and designed specialist architecture description languages (ADLs) to allow the precise definition of an architectural design. In reality however, in spite of a wide variety of such languages being available in the research domain, they are rarely applied to the definition of real information systems.

In this short paper, I explain the current state of practice, explain why current ADLs are not widely adopted by practitioners and suggest some requirements that software architects would have for new ADLs intended to address the information systems domain.

State of Practice

In my experience, information systems architects do not describe their architectural designs using languages that software architecture researchers would recognise as ADLs. The architects I know and work with use two very basic architectural description techniques: *UML* and *box-and-line* diagrams.

The *Unified Modelling Language* (UML) has fairly widespread recognition and many architects can read it (particularly version 1.x notations, version 2.0 notations are only now becoming more widely understood). Some can also use it fairly fluently to represent the kinds of models that they wish to create, although it is not necessarily the case that their audiences understand all of the nuances of their use of the language.

Particular strengths of UML as an ADL include its ubiquity, a generally good level tool support (at a wide range of prices and levels of sophistication) and a reasonably small, easily understood, core that can be used to represent much of the architecture of a mainstream information system.

However, UML has its limitations too. In particular, it is a very generic language containing few primitives that an information systems architect needs to use and so architects develop their own ways of using the language, which others may not understand. The lack of standard, widely understood, language extensions means that an architect has to choose between using a notation which results in their models looking familiar, but not communicating much of value, or introducing their own stereotype icons and running the risk of no one understanding that the model is actually represented in UML. Good examples of the former approach are the collections of guidelines assembled by Jeff Garland and Richard Anthony [1] and Nick Rozanski and myself [2].

The lack of direct support for representing common architectural constructs in UML leads many architects to conclude that it isn't worth the effort of using a standard language and instead, they develop their own notation (and perhaps semantics) using "*box-and-line*" diagrams or even new icons. A sophisticated example of such a special-purpose notation is Gregor Holpe's notation for enterprise application integration (EAI) systems introduced in [3], commonly known as "Gregorgrams". In effect, this is an information systems domain specific architecture description language, which specifically addresses the architecture description needs of one aspect of modern information systems development.

The advantages of using a specific language are clear: the language can be tailored to the task in hand and can be optimised for that task. With modern GUI tools like Eclipse, Visual Studio and Visio, editors can be created for graphical notations fairly easily and so the creation of diagrams using the new notations can be made efficient.

The problems with such specially created languages are also fairly clear: the architects creating them often don't have the time or expertise to create good notations with well defined semantics and not many people understand the notations when they are used (although to be fair, not many practitioners can read UniCon or xADL either).

In summary, the state of practice in architectural description for information systems, is fairly unsatisfactory with UML being the de-facto standard because of its wide visibility. However, most architects agree that it's not a very good language for architectural description (just the best they have) and so many resort back to "boxes-and-lines" to make themselves understood.

Existing Architecture Description Languages

Quite a number of purpose-designed ADLs appear to exist in the research domain, with just one source (the SEI ADL web page [4]) listing about 15, from AADL and ACME, to UniCon and Wright. Given that there are so many purpose designed ADLs, why are some of these languages not adopted by information systems architects? I would suggest that some of the main reasons are those outlined below.

- *Marketing.* In many cases, architects simply don't know that these languages exist and they are unlikely to learn about them unless they attend a specialist research conference, like ICSE, WADL or EWSA, which is unlikely given the subjects and attendance patterns of these conferences.
- *Priorities.* Most of the ADLs appear to focus on describing the functional and/or concurrency structure of the system. I haven't discovered one that places similar emphasis on information or deployment structure, both of which are key concerns for information systems architects.

- *Tool Support*. Most ADLs do not appear to have an associated software tool that a mainstream practitioner would feel comfortable using for their day-to-day work.
- *Representation*: Articles that describe the ADLs typically focus on formal and textual representations of the language, while many architects are used to graphical approaches.
- *Technical Mismatch*. None of the ADLs that I have read about represent modern information systems elements and constructs (such as message queues, publish/subscribe messaging, databases, web servers, application servers and so on) as first class language elements (or even standard extensions).
- *Perceived Relevance*. At present, there is little interaction between the practitioner and research communities and so practitioners are more likely to choose tools and approaches from commercial providers, as they will be perceived to be more relevant to their needs.

While it probably isn't necessary (or perhaps possible) to address all of these reasons for the lack of adoption, I feel that it will be necessary to address a reasonable proportion of them before an ADL is widely accepted by information systems architects. In the next section, we will review the priorities that an information systems architect would have for a new ADL.

Priorities of an Information Systems Architect

As alluded to in the previous section, I would suggest that a large part of the reason that existing ADLs have not been widely adopted for information systems stems from the differing priorities of the developers of the ADLs and the typical information systems architect. I would summarise the main requirements that an architect has for an architectural design notation as follows.

- *Support for Multiple Views*. Most approaches for information systems (and enterprise) architecture advocate the use of a number of views of the system, including functional, information, concurrency, deployment and so on. An information systems ADL will need to support the majority of the views that are commonly used by information systems architects.
- *Direct Domain Support*. Information systems architects want to be able to express their designs directly in terms of the standard types of system element that they work with in their domain. Concepts such as clients, servers, message queues of different sorts, component containers, data stores of different types, firewalls and networks of different sorts all need to be directly available as first class language constructs. Of course, this does not mean that these concepts need to be (or should be) part of the language core, but the language that architects actually use needs to include them as standard features.
- *Strong Tool Support*. The adoption of any textual or graphical notation is eased when powerful, usable tools are available to support it. To be a practical proposition, any new information systems ADL needs to be supported by suitable tools that will be familiar to the architect. In the current environment, this means making tool support available as extensions to familiar, existing toolsets such as Eclipse, Visual Studio and Visio.

- *Incremental Adoption*. It is unlikely that an organisation will be prepared to comprehensively adopt a new approach to architectural description in a single step. Successful adoption of a new ADL will involve architects successfully applying it incrementally to existing work and proving its effectiveness before applying it more widely.
- *Reuse of Models*. Creating a comprehensive architectural description is an involved and time-consuming process. An architect will find it much easier to justify investing this level of effort if the architectural models can be reused in a number of ways once created. Examples of such reuse could include automatic generation of skeleton systems (ideally with “round trip” facilities in the other direction) or the ability to perform performance analysis by applying a set of metrics to the model.

In summary, in order to attract wide use by architecture practitioners, a new ADL must be immediately usable in a way that will provide the architect with enough direct value from its use to justify the time and any direct costs of adoption.

Conclusions

There is much scope for improvement in the current state of practice of architectural description for information systems, in order to better support the architectural definition process.

Existing ADLs are not widely used by practicing information systems architects because, while undoubtedly rigorous, they do not align well with the specific needs and priorities of a practicing architect.

Should ADL designers wish to address the information systems domain, they must focus much more closely on the specifics of the domain and the practical, as well as conceptual, needs of those who will apply the languages developed.

References

[1] Jeff Garland and Richard Anthony, *Large-Scale Software Architecture*, Wiley, 2003.

[2] Nick Rozanski and Eoin Woods, *Software Systems Architecture: Viewpoint Oriented System Development*, Addison-Wesley, 2005.

[3] Gregor Holpe and Bobby Woolf, *Enterprise Integration Patterns*, Addison-Wesley 2003.

[4] Software Engineering Institute, *Architecture Description Languages* web page http://www.sei.cmu.edu/architecture/adl.html