The Merit of XML as an Architecture Description Language Meta-Language

Steve Pruitt, Doug Stuart, Wonhee Sull, and T.W. Cook
MCC (Microelectronics and Computer Technology Corp.)
Austin, TX, U.S.A.

Abstract

We investigated the use of XML, eXtensible Mark-up Language, as a way to represent software architecture models described with an Architectural Description Language (ADL). We believe XML provides a flexible and extensible framework highly suited to ADLs. Also, its adoption imparts many advantages that the ADL community can enlist to fulfil the responsibilities ADLs may soon be called to provide the industrial sector.

Introduction

The body of current ADL research contains a variety of dialects and implementations [LKA+95, AG97, Zel96, GMW97]. Most of the ADLs are focused within a particular research area and feature a corresponding set of capabilities. The advantages of a maturing ADL technology are increasingly sought after in the industrial sectors. Accordingly, we believe industrial adoption of ADLs will grow [Gar95]. Accompanying the increase in use will be an increase in the importance of software system architecture models within the enterprise. This increase in importance will result in expanded expectations for the versatility of software architecture models.

This paper is organized as follows. First, we list and describe a set of capacities necessary for a software-engineering environment. The capacities are examined with respect to the capabilities currently found in ADLs. Next, we describe the goals of our investigation. Following a brief overview of XML, we discuss how XML satisfies the list of capacities we identified as advantageous by a software-engineering environment.

Desirable SEE Capacities

The following list looks at the capacities modern enterprise development environments are likely to expect. While some of these capacities are intrinsically supported, or planned for, some of them will require additional ADL facilities.

1. **Representation** – This is the capacity to describe complex software architectures, an innate ability for any ADL. The ADL must feature a clear vocabulary for identifying and symbolizing the elements and the structure of the system [PW92]. The resulting model is required for the rest of the capabilities on the list.

2. **Analysis** – This is the capacity to evaluate architecture. Architecture evaluation includes, but is not limited to, dependency analysis, simulation, theorem proving, syntax checking, testability, reliability, etc [PW92].

3. **Traceability** – Requirements will inevitably change in the fast-paced environment that software systems are targeted toward. Two-way traceability between a set of requirements and the software architecture elements satisfying those requirements will be needed for impact analysis of changes at either end. In addition to requirements, the software architecture requires a flexible capability for associating system elements with other system artifacts, such as schedules, documentation, presentations, rationale, etc [PW92].

4. **Views** – Sharing understanding and building consensus is critical to the success of software systems [Med96]. Often a software system must be presented to a broad community with varying degrees of technical understanding and interest. A software system's architecture is the ideal representation for communicating; accordingly, the architecture must be able to support different views depending on the audience or message. The list of candidate views includes, but is not limited to, the components and services view, the process or execution view, the subsystem organization view, and the physical instantiation view [Kru95].

5. **Collaboration** – Increasingly, development is a distributed task. Software architecture will enable the distribution of its development geographically with either concurrent or independent development modes. Further, the development process must accommodate familiar and common tools for collaboration.

6. **Repository** – Software reuse is an anticipated capability arising from ADL adoption. Consequently, the ADLs must support the use of repositories, which are becoming critical infrastructure elements for software reuse.

Satisfying this range of capabilities requires interplay between the software architecture model and a wide set of applications and infrastructures. Therefore, a common standard for describing software architecture coupled with common parsers is required. Fortunately, there are conceptual similarities between most if not all of the ADLs making a standard description possible. We believe that XML provides that common standard and we have begun an investigation accordingly.

Goals

The initial goal of our investigation was determining the suitability of XML as a standard means to describe an ADL. Next, if XML was found suitable, our goal was to develop an XML schema for Acme, an ADL interchange language, and develop a simple prototype for creating the XML representation for an architecture described using Acme.

After successfully completing the first two goals, we have begun an effort to describe rationale capture artifacts in XML and then use the XML linking framework to establish relationships between a software architecture and its captured rational.

Brief XML Overview

XML is a proper subset of SGML, Standard Generalized Markup Language [Con96]. SGML is the international standard (ISO 8879) for defining descriptions of structure and content in electronic documents. SGML is a meta-language, a language for describing other markup languages. SGML is a very comprehensive standard and predates the widespread popularity of the WWW. It has long been noted that the WWW needed a semantic preserving data-description standard. XML is being developed, under the auspices of the World Wide Web Consortium, as a meta-language more suitable to the WWW informational needs.

The XML standard includes a set of associated standards. Foremost among them is XML Linking Language (XLink) and Document Object Model (DOM). XLink's facilities enable uni-directional and bi-directional relationships to be established between XML document objects [MD98]. DOM is a language-neutral interface giving applications programmatic access to the elements of an XML document [Wood+98]. XLink's status is working draft and DOM's status is recommendation.

As XML and its associated standards gain acceptance, there will be widespread commercial vendor support. Already, an impressive number of commercial document tool vendors have announced planned product versions supporting XML-based documents [Swen97]. This support coincides with the growing number of available commercial XML parsers that provide a consistent interface for manipulating XML documents [Gold98].

In the future, XML will garner additional support in the form of new applications and enhancements. The enhancements will likely include encryption and compression. Future applications will possibly include XML search engine support and navigation tools featuring 3-D environments.

XML Analysis

Many people associate XML with documents in the traditional sense. But XML's flexibility has the capacity to describe non-document-oriented information models [BPS98]. We investigated XML's capacity to describe ADL models and concluded that XML has several advantages as a standard representation language for ADLs.

A standard information description makes possible the broad sharing of ADL models so that many present and future applications can manipulate, search, present, and store the model. These applications will take advantage of access to a choice of COTS XML parsers [Rob97]. This trans-application capability enables ADL models to serve in multiple capacities.

Given the ongoing adoption of XML by industry, ADL models described in XML will be in a format that will not become orphaned [Rob97]. And, a standard open representation will de-couple an enterprise's architectural models from vendors and enable the models to remain useful despite the rapid change in software tools.

This section examines more closely the advantages of XML-described ADL models with respect to the aforementioned desirable SEE capacities.

1. **Representation -** The family of ADLs is somewhat large, but they all express themselves using a vocabulary of components, connections, and constraints [Med96]. These elements can be extended through properties and attributes and are organized by one or more relationship types. Serving as an ADL meta-language, XML has the clarity to describe an ADL consisting of these elements and their relationships. Further, XML's extensibility accommodates future changes to an ADL's definition [Bos97].

2. **Analysis** – Specific analysis techniques are accomplished by a using a specific ADL. Often system designers need to perform more than one analysis on their respective designs and they are faced with translating their design to the ADL featuring the desired analysis. Acme, an ADL interchange language, has been demonstrated to facilitate this task [GW98].

   ADL translation, through an interchange language or not, can be facilitated by using an open standard. Commonly available parsers can be selected and the same one can be employed for each ADL representation. The parsers will provide uniform interface to the underlying XML object model and translators can leverage the interface for source model access and target model creation.

3. **Traceablility** – Directly associating requirements with those architecture elements satisfying the requirements is important. If the requirements are captured in artifacts that can be addressed with an URL, then XML's linking facilities can point an architecture element to a system requirement artifact. If the requirements document supports the inclusion of URL, then the linking facilities can point a requirement to an architecture element. If the requirements document is an XML document then bi-directional links between requirements and architectural elements can be accomplished.

   Rationale capture tools are being adopted by software engineering environments and promise to be excellent candidates for directly linking with architectural models [Bose95].

   In addition to requirements and rationale, there are other types of artifacts that could benefit from direct links to the associated architectural elements. Additional artifact types include scenarios, test cases, engineering notes, or discussion groups.

4. **Views -** One of most important roles software architectures fulfill is communication. Through communication, the architect conveys the system intentions to peers and builds consensus. In doing so, the system architect must present views of the architecture, each featuring differing sets of information, for the intended audience. While the information for each view may overlap, there will be information distinct for a particular subset of views.

   XML's extensibility enables capturing the variety of information needed for supporting multiple architectural views [BPS98]. The information can be encapsulated within the architecture using XML tags, or linked to and from the architecture using XML's linking facilities. In either case, the information specific to one view can be made transparent to other views.

   Communicating differing views typically means the use of visual presentations, and graphical aids in general. This is one of the original intentions of XML. As an open standard, XML support in a variety of presentation tools will be common. The major commercial vendors of these tools are increasingly announcing support for XML.

5. **Collaboration** – The Internet is becoming the technology of choice for collaboration and XML is designed to be an Internet-compatible technology. Consequently, XML-based ADL applications would be able to leverage the existing volume of applications for presentation, browsing, and collaboration through the Internet and the WWW [PSL+98].

   In conjunction with Internet applications, XML's associated linking facilities will make it possible for an architectural model to have its development divided and then composed or linked together. This capacity will significantly support the collaborative needs of a distributed software-engineering environment.

6. **Repository -** Recent announcements have placed XML as the emerging standard for commercial repository vendors [Kril98]. Repositories are gaining acceptance as the primary means for storing and relating software system components and affiliated artifacts. It is certain that industrial adoption of ADLs will necessitate support for storing models in a repository, making the models accessible from repositories, and enabling extensive relationships between the software architecture elements and other relevant repository constituents.

Summary

XML is becoming the universal standard for the information description for the Internet. XML is extensible and flexible and we have shown it is an adequate way to describe an ADL model. XML would enable many of the ADL infrastructure needs to be accomplished using COTS. XML also positions ADL as a technology that can grow along with the rapid technological change expected in WWW technology. But, more importantly, XML is receiving the wide commercial support that is the most important criteria for industrial adoption of ADLs.

References

[AG97]. Formal Modeling and Analysis of the HLA RTI. Robert Allen, David Garlan. Proceedings of the 1997 Spring Simulation Interoperability Workshop, March 1997.

[Bos97]. XML, Java, and the future of the Web. Jon Bosak. Sun Microsystems, 1997.

[Bose95]. A Model for Decision Maintenance in the WinWin Collaboration Framework. Prasanta Bose. 10th KBSE Conference, 1995.

[BPS98]. Extensible Markup Language (XML) 1.0. http://www.w3.org/TR/1998/REC-xml-19980210. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen. W3, Febuary 1998.

[Con96]. XML Activity. http://www.w3.org/XML/Activity.html. Dan Connolly. W3, January 1996.

[Gar95]. Research Directions in Software Architecture. David Garlan. ACM Computing Surveys, Vol. 27, No. 2, June 1995.

[GMW97]. Acme: An Architecture Description Interchange Language. David Garlan, Robert T. Monroe, David Wile. Proceedings of CASCON '97, November 1997.

[Gold98]. The XML Handbook. Charles Goldfarb. Prentice Hall PTR, 1998.

[GW98]. A Case Study in Software Architecture Interchange. David Garlan, Zhenyu Wang. Submitted for Publication, March 1998.

[Kril98]. XML builds momentum as repository standard. http://www.infoworld.com/cgi-bin/displayStory.pl?980622.ehxml.htm. Paul Krill. InfoWorld, June 1998.

[Kru95]. The 4+1 View Model of Architecture. Philippe Kruchten. IEEE Software, November 1995.

[LKA+95]. David C. Luckham, John J. Kenney, Larry M. Augustin, James Vera, Doug Bryan, Walter Mann. IEEE Transactions on Software Engineering, Special Issue on Software Architecture, April 1995.

[MD98]. XML Linking Language (XLink). http://www.w3.org/TR/1998/WD-xlink-19980303. Eve Maler, Steve DeRose. W3, March 1998.

[Med96]. A Classification and Comparison Framework for Software Architecture Description Languages. Neno Medvidovic. Technical Report UCI-ICS-97-02, University of California, Irvine, February 1996.

[PSL+98]. Building XML Parsers for Microsoft's IE4. Jean Paoli, David Schach, Chris Lovett, Andrew Layman, Istvan Cseri. W3 Journal, 1998.

[PW92]. Foundations for the Study of Software Architecture. Dewayne E. Perry, Alexander L. Wolf. ACM SIGSOFT Notes Vol. 17, No. 4, Oct 1992.

[Rob97]. XML and Modern Software Architectures. SGML/XML 97, 1997.

[Swen97]. It's not just a .doc and .xls world anymore. John Swenson. MSDN Online. December 1997.

[Wood+98]. Document Object Model (DOM) Level 1 Specification. http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001. Lauren Wood+. W3, October 1998.

[Zel96]. The UniCon Language Reference Manual. Gregory Zelesnik. Carnegie Melon University, May 1996.