

Considerations on the Syntax of a Standard Exchange Format

Johannes Martin

Department of Computer Science
University of Victoria, Canada
Phone: +1-250-721-8788
email: jmartin@csr.csc.uvic.ca

Hausi Müller

Department of Computer Science
University of Victoria, Canada
Phone: +1-250-721-7630
email: hausii@csr.csc.uvic.ca

Abstract

The syntax of an exchange format for software engineering tools plays an important role in the success of the format. It should be easy to parse and emit, human readable, while at the same time powerful enough to express the most complex semantics that need to be modeled using these software engineering tools. This paper examines a number of exchange formats that are currently available with respect to these criteria.

Keywords: exchange format, syntax, comparison, CDIF, FAMIX, GraX, RDF, RSF, TA, XMI, XML

1 Introduction

The syntax of a software engineering exchange format primarily affects two tasks: *extraction* of information from software systems (parsing) and *visualization and processing* of this information. These tasks can be performed manually or with more or less sophisticated tools from simple Unix utilities to complex graph editors and specialized programming languages. Users include undergraduate students in a classroom setting, software engineers evaluating or occasionally using a tool, and power users that need to explore the full potential of their software engineering tools.

In order for a software engineering exchange format to support this wide range of users, it has to fulfill a number of requirements. To be used by software engineers and students without special tools it has to be *human readable* and its *core specification needs to be compact and easy* to understand. It should *not be restricted in its design* in a way that could prevent it from being used in contexts that are not anticipated at this time. This includes *scalability* and application to new programming paradigms or software engineering techniques. It should support *any level of detail* for the information to support modeling of both high level architecture and source code.

Moreover, it should *not enforce a specific order for related information*. For various reasons, it may not be feasible for a parser or tools to gather or hold the complete information on artifacts

in the software system investigated. Parsers have been written using simple tools such as *sed* or *awk* — they might have to emit partial information on an artifact at several times during the gathering of the information.

We mention the usability of the exchange formats in the classroom at various occasions throughout this document. In our opinion it is crucial for the success of the exchange format and software engineering tools in general that the tools be easy to use, so we can generate a positive attitude towards these tools in the software engineers of the future.

2 Examination of Current Exchange Formats

In the following paragraphs we will shortly describe the main features, advantages and shortcomings of various exchange formats that are currently being used in the software engineering community.

2.1 Non XML-based Exchange Formats

2.1.1 Rigi Standard Format (RSF)

The *Rigi Standard Format* (RSF) [12] was designed for the Rigi project [11]. It models information on software as typed, attributed, relational graphs. It is line oriented with every line defining a single node, arc, or attribute of these in a graph. This line orientation allows it to be used easily with standard Unix tools such as *grep* and *awk* for both parser generation and processing, filtering, and evaluation of parsing results.

A Rigi *domain definition* defines the valid node, arc, and attribute types for RSF files. The domain definitions are kept in separate files.

The original RSF did not allow for arc attributes or multiple arcs of same type, source, and destination. Several improvements to RSF have been made to solve these problems [10]. The syntax specification for the various RSF formats is short and simple, making it easy to implement tools using the RSF format.

2.1.2 Tuple Attribute Language (TA)

The *Tuple Attribute Language* (TA) [7] combines Rigi domain definition and RSF into one file. It allows for arc attributes, but, like the original RSF, does not support multiple arcs of same type, source, and destination. Its domain definition format is more flexible than that of Rigi in that it permits inheritance for node and arc types (such as: “a function is a procedure, and has a return type attribute in addition to the attributes of a procedure”).

TA is not line oriented, making it less accessible to Unix utilities and more difficult to parse. The

syntax, though more complex than that of RSF, is still easy to understand, and TA files can easily be read by humans.

Recent work has been done on wrapping TA into XML [8]. TAXML still does not allow for multiple arcs of same type, source, and destination.

2.1.3 CASE Data Interchange Format (CDIF)

The *CASE Data Interchange Format* (CDIF) [6] was developed by members of the Electronic Industries Association. Its specification is not freely available, which limits its chances for wide adoption by tool developers and researchers. CDIF does not seem to be maintained any more — the information on the CDIF web site is quite old and the cdif.org domain is no longer registered. Experiences gained in the development of CDIF are used in the design of XMI (see 2.2.3).

2.1.4 FAMOOS Information Exchange (FAMIX)

The *FAMOOS Information Exchange* (FAMIX) [4] was developed by the Software Composition Group at the University of Berne in an effort to point out and solve UML limitations with respect to round-trip engineering [3], in particular the lack of support for modeling implementation concepts in UML. Its specification is based on and compliant to the CDIF (2.1.3) standard. *Language Plugins* (roughly equivalent to Rigi domain definitions with notes on how to interpret them) and parsers for various object oriented languages (C++, Java, Smalltalk, Ada) exist. The current language plugins do not provide facilities for expressing very low level program artifacts, such as statements, and limit FAMIX' usefulness for applications such as program slicing.

2.2 XML-based Exchange Formats

2.2.1 Graph Exchange Format (GraX)

The *Graph eXchange format* (GraX) [5] was designed to be a powerful format for exchange of information between software reengineering tools. Like Rigi and TA, it uses the notion of graphs to represent information on software systems. In GraX, both nodes and arcs are first class objects, making it possible to attribute them. It also allows for multiple arcs with the same type, source, and destination. GraX introduces a total ordering on arcs, thereby facilitating the implementation of deterministic graph algorithms. A clear, short, and understandable overview of the language is available [5] and can be presented to undergraduate students as is.

2.2.2 Resource Description Framework (RDF)

The *Resource Description Framework* (RDF) [9] is a collaborative design effort by several W3C member companies. It emphasizes facilities to enable automated processing of Web resources [2].

RDF provides the means to describe arbitrary resources and their relationships, so it can be used to describe any structured data, such as software. It supports sets and sequences, similar to the total ordering facility in GraX. Since it is not targeted at software engineering applications, some work would have to be done to clarify in what ways RDF is to be used in this context. The RDF specification is quite compact so it can be used in the classroom, though a compact document describing RDF and its role as a software engineering exchange format would be desirable.

2.2.3 XML Metadata Interchange (XMI)

The *XML Metadata Interchange* (XMI) [1] has originally been designed as an XML compliant version of the *Unified Modeling Language* (UML), a specification of the *Object Management Group* (OMG) that provides application developers a common language for specifying, visualizing, constructing, and documenting distributed objects and business models, so it is primarily targeted at modeling mid and high level software artifacts. Before XMI can be used as a common interchange format in our community, work will have to be done on finding a consistent way to describe all levels of software artifacts. The specification documents for XMI are very large and unsuitable for use in the classroom; more compact documents describing both XMI and its usage for software engineering would have to be written.

3 Conclusion and Position

Several of the current exchange formats satisfy the technical requirements of a common interchange format. XML, though very verbose, is to be preferred as the base format because of its wide acceptance as general data exchange format on the Internet and the resulting availability of generic XML tools.

Because of its complexity, XMI does not appear to be a good choice for our chronically understaffed community. Support for RDF should be more manageable, and tools developed for RDF might prove useful for our community, but its specification is still quite large. Since RDF is not targeted at describing software but arbitrary metadata and resources, some work would have to be done to define in what way this formats is to be used in the software engineering tools context, possibly by defining a specialization of it.

XML versions of RSF or TA may be considered, if they solve the deficiencies of their non-XML counterparts as described in the previous paragraphs. GraX supports all features of RSF and TA, is powerful and flexible enough to accommodate for most, if not all, usage scenarios, can be used in the classroom, and solves many of the problems associated with traditional RSF and TA. It therefore seems a good choice for a common exchange format. Upgrading tools that support TA or RSF to support GraX should be a very manageable task. We should take a close look at GraX, identify and solve potential problems, and accept it or an improved version of it as our common exchange format.

References

- [1] XML Metadata Interchange (XMI). Technical report, Object Management Group, October 1998. <ftp://ftp.omg.org/pub/docs/ad/98-10-05.pdf>.
- [2] Tim Berners-Lee and Ralph R. Swick. Frequently Asked Questions about RDF. Technical report, World Wide Web Consortium, September 1999. <http://www.w3.org/RDF/FAQ>.
- [3] Serge Demeyer, Stéphane Ducasse, and Sander Tichelaar. Why FAMIX and not UML? In *UML'99 Conference Proceedings*, LNCS. Springer Verlag, 1999. <http://www.iam.unibe.ch/~famoos/FAMIX/whyFAMIX/whyFAMIX.html>.
- [4] Serge Demeyer, Sander Tichelaar, and Patrick Steyaert. FAMIX 2.0 — The FAMOOS Information Exchange Model. Technical report, Software Composition Group, University of Berne, August 1999. <http://www.iam.unibe.ch/~famoos/FAMIX/Famix20/Html/famix20.html>.
- [5] Jürgen Ebert, Bernt Kullbach, and Andreas Winter. GraX - An Interchange Format for Reengineering Tools. Fachberichte Informatik 5/99, Universität Koblenz-Landau, Institut für Informatik, 1999.
- [6] Johannes Ernst. Introduction to CDIF. Electronic Industries Association, September 1997. <http://www.eigroup.org/cdif/intro.html>.
- [7] Ric Holt. An Introduction to TA: The Tuple-Attribute Language. Technical report, University of Waterloo and Toronto, February 1997. <http://plg.uwaterloo.ca/~holt/papers/ta.html>.
- [8] Ric Holt. TA in XML. Technical report, University of Waterloo and Toronto, December 1999. <http://plg2.math.uwaterloo.ca/~holt/taxml/>.
- [9] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Recommendation, World Wide Web Consortium, February 1999. <http://www.w3.org/TR/REC-rdf-syntax/>.
- [10] Johannes Martin. RSF file format. Technical report, University of Victoria, August 1999. <http://www.rigi.csc.uvic.ca/list-archives/rigi-developer-archive/2000-02-10-15:26:16-19165>.
- [11] H. A. Müller and K. Klashinsky. Rigi — A system for programming-in-the-large. In *Proceedings of the 10th International Conference on Software Engineering*, pages 80–86, 1988.
- [12] Hausi A. Müller, Mehmet A. Orgun, Scott R. Tilley, and James S. Uhl. A reverse engineering approach to subsystem structure identification. *Journal of Software Maintenance: Research and Practice*, 5(4):181–204, December 1993. <ftp://ftp.rigi.csc.uvic.ca/pub/papers/jsm.ps.Z>.