ACM SIGSOFT Software Engineering Notes, Volume 26, January, 2001, pages 44-49.

# WoSEF: Workshop on Standard Exchange Format

Susan Elliott Sim
University of Toronto
simsuz@cs.utoronto.ca

Rainer Koschke
University of Stuttgart
koschke@informatik.uni-stuttgart.de

## Abstract

A workshop was held at ICSE 2000 in Limerick, Ireland to further efforts in the development of a standard exchange format (SEF) for data extracted from and about source code. WoSEF (Workshop on Standard Exchange Format) brought together people with expertise in a variety of formats, such as RSF, TA, GraX, FAMIX, XML, and XMI, from across the software engineering discipline. We had five sessions consisting of a presentation and discussion period and a working session with three subgroups. The five sessions were: 1) Survey and Overview, 2) Language-level schemas and APIs, 3) High-level schemas, 4) MOF/XMI/UML and CDIF, and 5) Meta schemas and Typed Graphs. During that time we reviewed previous work and debated a number of important issues. This report includes descriptions of the presentations made during these sessions. The main result of the workshop is the agreement of the majority of participants to work on refining GXL (Graph eXchange Language) to be the SEF. GXL is an XML-based notation that uses attributed, typed graphs as a conceptual data model. It is currently a work in progress with contributors from reverse engineering and graph transformation communities in multiple countries. There is a great deal of work to be done to finalise the syntax and to establish reference models for schemas. Anyone interested is welcome to join the effort and instructions on how to get involved are found at the end of the workshop report. Three papers from the workshop have been reprinted here to promote reflection and encourage participation in the work to develop an SEF.

## Introduction

Researchers in computer-aided software engineering tools and computer-aided reverse engineering tools have recognized a standard exchange format (SEF) as a means for improving the state of the art of tool interoperability. As part of an on-going effort, the Workshop on Standard Exchange Format (WoSEF) was held at ICSE2000 on Tuesday, 6 June, 2000. The workshop was co-chaired by Susan Elliott Sim, Rainer Koschke, and Richard C. Holt. This meeting was primarily concerned with moving towards consensus on a common format for sharing data about source code. These data could be extracted from source code using parsers, inferred using analysis tools, or taken from other sources such as configuration management systems. There is a multiplicity of reasons to have an SEF. It would allow us to use a best of breed approach when selecting tools and it would avoid unnecessary duplication when writing tools. This motivation was reflected in the reasons that the participants gave for attending the workshop: 1) They wanted to be able to make complementary tools work together more smoothly; and 2) They were tired of writing parsers/analyzers and wanted to avoid writing another one, in particular a C++ parser. A small number of participants were attending because they were developing a format for use within their tool sets and wished to see successful examples. Aside from the benefits for tools, an SEF would enable the creation of benchmark problems, or "guinea pigs," for a particular area. Such guinea pigs would serve to codify knowledge about tools, a class of problems, and problem solving techniques.

There have been many prior efforts to create an SEF. Some of these are general-purpose exchange formats that can be adapted to data about software, while others are specifically for software. Some examples are XML (eXtensible Mark-up Language)[3], with a specialized form, XMI (XML Metadata Interchange format) [18], RDF (Resource Descriptor Format)[9], RSF (Rigi Standard Form)[10], TA (Tuple Attribute Language)[14], GraX[2], and CDIF (CASE Data Interchange Format)[1]. These formats vary in the amount of support and use they receive. This proliferation of exchange formats underlines both the need for a standard format and the lack of consensus on one.

For the first time, this workshop brought together members from disparate communities to share experiences and work together. The workshop summarized previous work in this field and re-visited XML, XMI, UML, and CDIF. We discussed concrete schemas for high-level information, such as class diagrams or architectural information, and for low-level information, such as abstract syntax trees. We also debated the concepts and the mechanisms to specify meta schemas. The interaction between the participants were lively and the discussion sessions often ran overtime (to the great chagrin of one of the co-chairs). On occasion, someone would jump up to illustrate a point on the flip chart. During the breaks, small groups would form in front of the blackboard to pursue or clarify arguments.

We accepted 13 position papers and these fell into five groups, which we used to organize the workshop. We scheduled five sessions, each with a presentation followed by discussion, on the following topics:

1. Survey and Overview
2. Language-level schemas and APIs
3. High-level schemas
4. MOF/XMI/UML and CDIF
5. Meta schemas and Typed Graphs

To maximise time for discussion, we appointed a chair for each session to co-ordinate a combined presentation for each group of papers. We found this format was very effective for identifying commonalities and relative strengths of the various approaches. We recommend this organization to other workshop organisers. These sessions will be described in Section 2.

During the last two hours of the workshop, we departed from the planned presentations and discussions. For one and a half hours, we broke into three small groups, each focussing on a single topic. These topics were suggested by discussions earlier in the day, and

[1] Work on CDIF has been transferred to the XMI effort.

they were i) high-level schemas, ii) C++ schema and API, and iii) notation for exchanging schemas. The groups identified requirements, made prescriptions for progress, and wrote will-do lists. Afterwards, each group presented its results to the whole workshop. Their work is described in Section 3.

By the end of the of the day, this workshop produced two major results:.

- Virtually all participants committed to refining GXL (Graph eXchange Language) a format in-progress. GXL uses XML syntax and is used to encode attributed graphs. The groups who committed to working together further were University of Stuttgart, Bell Canada, IBM Canada Ltd., Mahindra-British Telecom, University of Waterloo, University of Koblenz, University of the German Federal Armed Forces (Munich), Philips Research Eindhoven, University of Victoria, and Nokia.
- We identified three major areas for continued work, coinciding with working groups. Web pages and mailing lists have since been created for these topics.

These results, specifically GXL, will be discussed in Section 4. Finally, we give instructions on how to become involved in the effort to establish an SEF in Section 5.

# Sessions

As mentioned above, we had five sessions of presentations and discussions. Each of the sessions was organized according to common themes in the position paper. We began with a kick-off presentation by Ric Holt to establish a common vocabulary. The first session, "Survey and Overview," reviewed previous work on SEFs. Subsequent sessions addressed successively higher level issues. They covered "Language-level schemas and APIs," "High-level schemas," "MOF/XMI/UML and CDIF," and "Meta schemas and Typed Graphs." Summaries of these presentations are presented in this section of the workshop report.

## *1.1 Survey and Overview*

Holger Kienle gave the joint presentation for this topic covering papers by Kienle, Czeranski, and Eisenbarth; Martin and Muller; and Riva [13][16]. The presentation covered background material and previous work on exchange formats.

There are many different published exchange formats. Some of them were explicitly designed to be generally usable, others originated in related areas and could easily be adapted. Among the formats explicitly designed as general exchange formats, one can identify two major subclasses: those invented in a software engineering context (e.g., RSF, TA, CDIF, and GraX) and those provided by other communities and nevertheless useful to exchange software artifacts (e.g., ASN.1, XDR, Abstract Syntax Description Language, Resource Description Framework, XML). Some examples of the latter have evolved in the graph drawing community and the compiler community. Because of the rich expressiveness of graphs, some researchers have found the graph data model relevant to software engineering. The compiler community has also evolved several persistent intermediate representations that can be used to transfer low-level program information.

There are two different groups of different stakeholders for exchange formats: users of the format and tool builders. Not surprisingly, the requirements of these two groups differ. For example, from the user's point of view, the exchange format should be human readable, compact, context-neutral, and extensible, while a tool builder wants an interchange format that can easily and efficiently be parsed, stored, and generated.

The group at Nokia Research represents both view points when they describe their needs for an exchange format to be used as part of their reverse architecture process. The process consists of three consecutive major steps: extraction, abstraction, and analysis of software architectures. For each step, different tools need to be used because there is currently no tool available that supports the whole process. Hence, the ability to easily exchange information among these tools is crucial.

## *1.2 Language-level schemas and APIs*

Any tool that parses source code, such as analyzers and compilers, uses some internal or intermediate representation of the source with an underlying schema. The schema of the internal representation might not necessarily be explicitly documented (in any other form than the code), neither may the internal representation be designed for more than temporary data storage, but both schema and the raw data exist nonetheless. Language-level schemas are used to define the structure of programming representations for either specific languages or a family of related languages, e.g., procedural languages. The representations are often accompanied by an API (application program interface) to access the data. The programming language Ada even has a standardized API, called Ada Semantic Interface Specification (ASIS) that several compiler vendors support[6]. In some other sub-communities, a particular front end defines a de-facto standard. For example, the Datrix group at Bell Canada provides members of CSER (Consortium for Software Engineering Research), a group of Canadian researchers and companies involved in reverse engineering research, with a C++ front end that emits annotated abstract syntax trees. The presentation for this topic was given by Sébastien Lapierre and covered two papers, one by Lapierre, Laguë, and Leduc and a second one by Kienle, Czeranski, and Eisenbarth[13][16].

Lapierre, a member of the Datrix group, reported that use of the Datrix front end ranges from metric computation, clone and design pattern detection to program slicing and refactoring. For these analyses, it is also necessary to allow read as well as write operations to the AST. Moreover, a user of the intermediate representation should also be able to add higher-level elements, as for example, architectural concepts, and additional abstract links. These annotations are, for instance, needed to record what code pieces have been identified as clones of each other. Lapierre observed that it is currently not easy to benefit from other researcher's tools and noted that there are basically three technical ways to ease a closer collaboration. One is to unite all analyzers into one huge open source project, another one is to provide standardized exchange formats, or – finally – one can achieve tool interoperability by means of APIs.

The challenges in API design are:

- to find a suitable way for returning or giving access to the result of a query (should the same model be enhanced or should parallel models/copies be created?),
- to find mechanisms to annotate or colour AST nodes,
- to provide means to modify the AST (without compromising AST integrity), and
- to offer state management mechanisms to permit undo, roll-back, and reset operations.

Further points to ponder are whether access to the AST should be embedded into a high-level programming language or rather be by means of a domain-specific query language, and the question of how different tools (visualization, graph manipulations, etc.) can be integrated without duplicating the data.

### *1.3 High-level schemas*

Language analyzers, like compilers, extract very detailed information from source code. The amount of extracted data can be far too large to be comprehended or analysed in a reasonable amount of time. For many reverse engineering activities, only a broad overview of the system is necessary. Typically, only global code entities, such as functions, global variables, user-defined types, and classes together with their relationships are really relevant for a broad overview. Whereas expressions and statements in the body of functions can be used to induce certain relationships among code entities but can otherwise be ignored. Such high-level information can then be visualized, analyzed, and manipulated. In contrast to forward engineering, high-level information is usually the starting point of the design. A class diagram in UML, for example, does not contain individual statements, but classes and their relationships.

Similar to low-level extractions, every high-level description of system, implemented or not, has an underlying high-level schema. Michael Godfrey from the University of Waterloo summarized papers by Neuhold, and Hess and Schulz and a portion of his own paper [13][16]. He distinguished between two different levels of abstraction in high-level schemas: programming language entity level (globally declared code entities directly derived from a system) and architectural level (components and connectors). For the programming language entity level, he presented several high-level schemas for procedural and for object-oriented programs independently developed by different organizations and their corresponding extractors. Interestingly enough, these schemas were quite similar, which suggests that reaching an agreement on a common high-level schema for a particular language or paradigm is a realistic goal.

### *1.4 MOF/XMI/UML and CDIF*

The presentation by Louis St-Pierre summarized papers by St-Pierre, Tichelaar et al., and Dirckze et al. [13][16] on a family of related standard exchange formats widely used in academia and industry, namely, MOF (Meta Object Facility) [8], XMI [18], UML (Unified Modeling Language) [11], XML [3], and CDIF. St-Pierre began by revisiting the three levels of the Object Modeling Group (OMG) model-driven approach:

- M1: concrete models that describe application models, e.g., a finance application model
- M2: metamodeling languages provide syntax and semantics to describe concrete models; a specification using a metamodeling language is called a metamodel; e.g., UML or FAMIX can be used to describe a concrete finance application model
- M3: meta-metamodeling languges provide syntax and semantics to describe M2 models, e.g., MOF is used to specify UML; meta-metamodeling languages can also be used to define rules for metadata interchange format generation (e.g., XMI format including XML DTDs for content verification)

St-Pierre noted that the M3 level is needed for abstraction (because the model abstracts functionality from implementation) and to enable semantic interoperability. After explaining the relationships among UML, MOF, XML, and XMI and reporting on Tichelaar et al.'s experiences with CDIF, he concluded that MOF and XMI are our best options at the moment because they are accepted standards and tool support is already available, while CDIF has been virtually abandoned. However, UML does not adequately represent some features of source code, in particular of procedural programs. UML forces certain interpretations (e.g., the distinction between inheritance used as generalization or as implementation re-use) while other information is beyond the scope of UML, for instance, PICTURE clauses in COBOL.

### *1.5 Meta schemas and Typed Graphs*

This session built on the previous one by considering the M2-level schema for an interchange format, or meta schema. This discussion was primarily concerned with the schema of the format rather than the schema of the data that it encodes. The papers by van den Brand et al., Godfrey, and Ebert et al. [13][16]described formats that have different meta schemas, specifically ATerms and Tgraphs.

Rather than tackling these meta schemas directly, the presenter, Andreas Winter, identified five components of an SEF and explained how approaches from the three papers fell into the taxonomy. The five components were: 1)structure, 2)format, 3)meta schema, 4) data access, and 5) transformation. "Structure" is the underlying conceptual data model of the SEF, such as ASTs, graphs, and relational models. "Format" is the form in which the data is transferred. "Meta schema" is the notation for representing the schema of the data for transmission between tools. The fourth component is a standardised mechanism for accessing the stored data efficiently, such as an API or set of utilities. "Transformation" is the capability to transform data from one schema to another.

After using this taxonomy to compare the three approaches, Winter made the following conclusions. The SEF would need to encode both schema and instance data to maximise portability between tools. The SEF should use directed, attributed, typed graphs as the underlying structure and be based on a common meta model. He also recommended that the SEF should use XML syntax to leverage existing tools.

## Working Groups

Following the afternoon break, the participants separated into small groups to discuss specific topics. These issues evolved out of the discussions during the earlier sessions as points of

disagreement or unresolved problems. Each group was instructed to work on three items: 1) requirements for solving the problem; 2) a prescription for making progress, and 3) "will-do" lists. We asked for "will-do" lists instead of "to-do" lists, because we wanted concrete action items. Again, the discussions were lively and participants learned a lot from each other. These working groups were a high point in the day, despite being a last-minute change to the schedule.

Following one and a half hours of discussion, the workshop reconvened in a single group. Each working group selected one person to present its results. In the remainder of this section, these presentations will be summarized.

### *1.6 High-level Schemas*

The participants in this working group were familiar with a large number of high-level schemas and they quickly realised there was not enough time to identify specific requirements that were common to all of them. Instead, they added this task to the will-do list and concentrated on broad requirements. Since there was a lot of agreement in the group, they were able to make good progress in their discussions.

**Requirements**

- Design schemas for a variety of "high" levels, including (but not limited to) one for language-level and one for architectural level.
- These schemas should support the needs of tool developers.
- These schemas must work with C/C++ and Java.

**Prescription for Progress**

- Impose one or more standard schemas and later correct them based on feedback.
- Architecture level schema may be straightforward and should be considered as first candidate.
- Somebody (not us!) should also consider "software architecture" in the theorem proving sense, i.e. specifying the low-level semantics of interactions between components using formal logics.

**Will-Do List**

- Join an email list to continue the discussion.
- Explore "Guinness-enabled reverse engineering" later tonight

Volunteer to participate in the validation of a standard schema.

### *1.7 C++ schema and API*

There was a strong desire among WoSEF participants for a robust and flexible C++ parser. The group recognised that one way to achieve the goal was to place data parsed from source code into a repository and use an API to access the data. Consequently, this working group considered the problem of what data should be placed in the repository and how the corresponding API should work. The discussions in this group was particularly complex and contentious, and the group was the last to re-join the main group.

**Requirements**

- The API must be able to traverse, query, transform graphs or trees.
- The API must be connected to a high-level programming language.

**Prescription for Progress**

Examine existing APIs for C++ representations, in particular, Datrix, IBM Visual C++ CodeStore, and g++, and create a single unified schema.

### *1.8 Notation for exchanging schemas*

This topic had the smallest working group and it spent most of its time defining terms from which discussion could proceed. The group explored a number of issues and options and learned more about what didn't work than what did work. When they reported back to the workshop, they gave only a will-do list.

**Will-Do List**

- Define a notation for representing schemas that can be used with GXL. This notation will likely be a subset of UML class diagrams.
- Collect a number of example schemas. This collection will be used to guide the design of the notation.
- Implement converters between GXL and the following formats: RSF, TA, TGraphs, PROGRES, RPA, and FAMIX.

## Results

Beyond gathering momentum and stimulating discussions on diverse issues on exchange formats, the major result of the workshop was the commitment of the workshop participants to work on GXL as a common exchange format. Since GXL is a promising candidate for a standard format for transferring data on software artefacts, we will now use the opportunity to briefly present GXL.

GXL allows to encode typed, attributed, multi-graphs with edges as first-order entities (edges themselves can have attributes). It originates from a number of graph-based exchange formats, GraX (University of Koblenz-Landau) [1], TA (University of Waterloo) [14], and the graph format of the PROGRES graph rewriting system (University of the German Federal Armed Forces, Munich) [12]. Furthermore, GXL includes concepts from the exchange format of Relation Partition Algebra, RPA (Philips Research Eindhoven, The Netherlands) [5] and RSF (University of Victoria, Canada) [10].

GXL is an XML sublanguage and its exact syntax specification is still subject to an ongoing discussion. The following is an excerpt of version 0.6.6, released 31 August, 2000. The latest version of the XML Document Type Definition (DTD) can be found at http://www.gupro.de/GXL/.

```
<!ELEMENT gxl (graph)* >
<!ELEMENT graph (attr* , ( node | edge | rel )*)
>
<!ATTLIST graph
    id          ID       #REQUIRED
    schema      CDATA    #IMPLIED
    edgeids     ( true | false ) "false"
    undirected  ( true | false ) "false"
    hypergraph  ( true | false ) "false"
>
<!ELEMENT node (attr*, order*) >
<!ATTLIST node
    id      ID       #REQUIRED
    type    NMTOKEN  #IMPLIED
>
<!ELEMENT edge (attr)* >
<!ATTLIST edge
    id     ID       #IMPLIED
```

```
    type   NMTOKEN  #IMPLIED
    from   IDREF    #REQUIRED
    to     IDREF    #REQUIRED
>
<!ELEMENT attr ( (%type;)?, (%val;)? , attr*) >
<!ATTLIST attr
    name   NMTOKEN  #REQUIRED
    kind   NMTOKEN  #IMPLIED
>
```

The following interpretation of the DTD deliberately omits some details for explanatory purposes. In GXL, graphs consist of nodes, and edges.[2] All of these entities, graphs, nodes, and edges, must have unique identifiers and may have attributes. In the case of edges, this unique identifier may be given explicitly or it may be implied by the end points and type of the edge. One of these two options must be selected for the entire graph by setting the graph attribute "edgeids" appropriately. A similar choice must be made for whether the graph has directed or undirected edges. Attributes defined by giving a name and a value. This value must have a type, either a primitive type (int, float, boolean, string) or a composite type (bag, set, seq, struct).

The syntax for representing schemas is currently under discussion, but it will also use a graph-based representation. As a result, schemas can in turn be exchanged as graphs represented within the GXL document. Consequently, both instance data and schema data can be transferred.

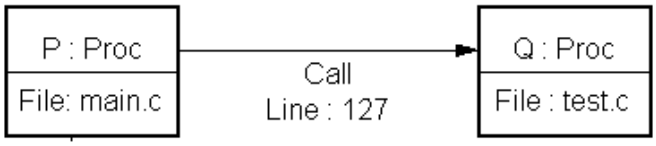


**Figure 1: Graph Representation of Source Code**

To give a flavour of how GXL encoded data look like, consider the following information to be represented with GXL: procedure P (declared in main.c) calls procedure Q (declared in test.c) in line 127. This information is represented graphically in Figure 1. The GXL document for this example is as follows:

```
<?xml version="1.0"?>
<!DOCTYPE gxl SYSTEM
"http://www.gupro.de/GXL/gxl.0.5.dtd">
<gxl schema         =
"http://www.gupro.de/GXL/demo.xml"
     edgeid = "false"
     undirected = "false"
     hypergraph = "false"
>
<node id = "P" type = "Proc">
  <attr name = "File">
    <string> main.c </string>
  </attr>
</node>
<node id = "Q" type = "Proc">
  <attr name = "File">
    <string> test.c </string>
  </attr>
</node>
<edge id = "c" type = "Call" begin = "P" end =
"Q">
  <attr name = "Line">
    <int> 127 </int>
  </attr>
</edge>
</gxl>
```

Like other XML sublanguages, GXL is quite verbose. In order to exchange large amount of data, for example, abstract syntax trees for larger systems, standard compression methods need to be used to save space. On the other hand, existing XML tools can be used to traverse the graphs and access the data they encode.

## Future Work

Since WoSEF was held, other workshops have been organized to develop a standard exchange format. There will be a meeting at APPLIGRAPH (Applications of Graph Transformation subgroup) [1] to be held 5-6 September, 2000 in Paderborn University in Germany to find an XML-based SEF for graph transformation systems. The graph drawing community will hold a kick-off meeting for an initiative to find an SEF on 20 September, 2000 as part of at their annual meeting, GD 2000 in Colonial Williamsburg, USA [7]. There will be a panel on standard exchange formats at WCRE 2000 (Working Conference on Reverse Engineering) [17] in Brisbane, Australia, 23-25 November, 2000. Finally, a Schloss Dagstuhl seminar is planned for January, 2001 [4].

As is evident from these meetings and this workshop report, work on a standard exchange format is far from complete. A great deal of work remains to be done and anyone is welcome to join the effort. There are a number of ways to learn more about current work and get involved:

- Visit the workshop home page at http://www.cs.utoronto.ca/~simsuz/wosef for details of ongoing work.
- Contact any of the authors of this paper for more information.
- Join the Waikiki Beach Club mailing list for email updates on the work. Instructions are available at: http://www.informatik.uni-stuttgart.de/ifi/ps/waikiki [15]
- Read the position papers from WoSEF for background information.

To help you get started, three position papers from the workshop have been selected and included in this issue of Software Engineering Notes that together cover a broad spectrum of issues in interchange formats. While the first paper gives a bibliographic background for existing interchange formats, the other two papers concentrate on specific program representations for the lower-level entities and higher-level entities.

The first paper by Kienle, Czeranski, and Eisenbarth of the University of Stuttgart reviews and classifies existing exchange formats. Both domain-specific (e.g., interchange formats for graph drawing tools) and general-purpose formats (e.g., XML) are

[2] A "rel" is a hyperedge, that is, an edge with more than two endpoints, such as those used in some entity-relationship diagrams and class hierarchies. Hyperedges can only occur in hypergraphs, and the corresponding attribute of the graph is set to true.

discussed. Included in the survey are persistent intermediate representations for compilers as candidates for low-level program representations. Finally, advantages and disadvantages of persistent data structures as alternatives to exchange formats are debated.

The Datrix group at Bell Canada has considerable experience in providing members of CSER with program extractions in the shape of annotated abstract syntax trees for C++. The Datrix group members Lapierre, Laguë, and Leduc report on their experience with selecting Datrix-TA, a variant of TA, to represent these annotated abstract syntax trees, called abstract syntax graphs by the authors.

Godfrey of the University of Waterloo concentrates on high-level program schemas suitable to represent necessary information for architecture recovery. His experiences in using different extractors that generate high-level program schemas along with a detailed list of requirements for such schemas are described in his paper. Furthermore, he points out several important practical problems with uniquely identifying and resolving entities when different extractions are linked together to a global system representation (especially if the separate representations stem from different extractors) and how these entities need to be tracked back to their original source.

## Attendees

Participants: Marat Boshernitsan, Rahul Charmadhikan, Raj Chittar, Mike Godfrey, Hoh In, Holger Kienle, Kostas Kontogiannis, Bernt Kullbach, Sébastien Lapierre, Tim Lethbridge, Johannes Martin, Hausi Müller, Karin Neuhold, Stephen Perelgut, Derek Rayside, Claudio Riva, Tobias Rötschke, Louis St-Pierre, Sander Tichelaar, Andreas Winter, Wai-Ming Wong

## References


[1] APPLIGRAPH Subgroup Meeting on Exchange Formats for Graph Transformation. http://www.uni-paderborn.de/cs/ag-engels/Conferences/APPLIGRAPH_XML/, 1 September, 2000.

[2] Jürgen Ebert, Bernt Kullbach, and Andreas Winter. “GraX—An Interchange Format for Reengineering Tools” Proceedings of the Sixth Working Conference on Reverse Engineering, pp. 89-98, Atlanta, GA, 6-8 October, 2000, Los Alamitos: IEEE Computer Society Press.

[3] Extensible Markup Language Home Page. http://www.w3.org/XML/, 1 September, 2000.

[4] Dagstuhl Seminar 01041, Interoperability of Reengineering Tools, http://www.dagstuhl.de/DATA/Reports/01041/, 1 September, 2000.

[5] L. M. G. Feijs and R. C. van Ommering. “Relation partition algebra – mathematical aspects of uses and part-of relations,” Science of Computer Programming, 33(2), pp. 163-212, February 1999.

[6] International Standards Organization. ISO/IEC 15291 Ada Semantic Interface Specification (ASIS), 1999.

[7] Graph Drawing 2000 home page, http://www.cs.virginia.edu/~gd2000/, 1 September, 2000.

[8] MOF Revision Task Force home page. http://www.dstc.edu.au/Research/Projects/MOF/rtf/index.html 1 September, 2000.

[9] RDF (Resource Descriptor Format.) http://www.w3.org/RDF/, 1 September, 2000.

[10] RSF (Rigi Standard Form) http://www.rigi.csc.uvic.ca/rigi/manual/user.html, 1 September, 2000.

[11] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*, Addison-Wesley Publishing Company, 1998.

[12] Andy Schürr. “Developing Graphical (Software Engineering) Tools with PROGRES , Formal Demonstration,” in Proceedings of the Nineteenth International Conference on Software Engineering (ICSE'97), pp. 618-619, Boston, Massachusetts, 18.-23. May 1997, Los Alamitos: IEEE Computer Society Press.

[13] Susan Elliott Sim, Richard C. Holt, Rainer Koschke. “Workshop on Standard Exchange Format Proceedings.” 6 June, 2000, Twenty-Second International Conference on Software Engineering, Limerick, Ireland.

[14] Tuple Attribute Language. http://plg.uwaterloo.ca/~holt/papers/ta.html, 1 September, 2000.

[15] Waikiki Beach Club Home Page http://www.informatik.uni-stuttgart.de/ifi/ps/waikiki, 1 September, 2000.

[16] WoSEF (Workshop on Standard Exchange Format) Home Page. http://www.cs.utoronto.ca/~simsuz/wosef, 1 September, 2000.

[17] Working Conference on Reverse Engineering (WCRE), 2000 home page. http://www.reengineer.org/~wcre2k, 1 September, 2000.

[18] XMI (XML Metadata Interchange Format). http://www.software.ibm.com/ad/features/xmi.html, 1 September, 2000.