Defining, Transforming, and Exchanging High-Level Schemas

Michael W. Godfrey
Software Architecture Group (SWAG)
Department of Computer Science
University of Waterloo
Waterloo, Ontario, CANADA
email: `migod@acm.org`

Abstract

Recently, the reverse engineering research community has been investigating mechanisms for exchanging data and partial results between reverse engineering tools [10, 7]. Among the many subproblems inherent in implementing a standard exchange format (SEF) is the design of schemas that represent views of source code at various levels of abstraction. In this paper, we briefly discuss some of the issues in defining, transforming, and exchanging "high-level" schemas, including previous work on the TAXFORM project [2] as well as recent discussions at the Workshop on Standard Exchange Format (WoSEF) [7].

1 Introduction

To be able to exchange "facts" or other information gathered from diverse reverse engineering tools, several thorny issues must be addressed [2, 4]:

1. An underlying *transport mechanism* must be agreed upon. This includes not only deciding upon a concrete syntax but also a lowest (common) level of abstraction.

2. A set of underlying *semantic models* (data schemas) must be created. These schemas must be capable of representing software systems written in different programming languages at different levels of detail, including the AST/ASG level, the program entity (external declaration) level, and the architectural level.

3. Various *implementation issues* must be addressed, such as devising a convention for generating unique identifiers (UIDs) for program entities. We note that it is not our intention to discuss these kinds of implementation issues here as we have done so elsewhere [2].

As to the choice of a concrete syntax, we note that there are several attractive options including variants of UML, the TA language (which is based on a simple entity-relation model using textual tuples and attributes, such as `calls f g`) [6], and GXL (a notation for encoding information about directed graphs using XML) [8].

The remainder of this paper discusses the design of high-level data schemas.

2 High-Level Schemas

Previous discussion of schemas for representing facts about software systems has concentrated on three levels of detail [10, 9, 7]:

1. The *AST/ASG* level, which models information at the program statement level (*e.g.,* local variables and low-level control flow logic).

2. The *program entity* (external declaration) level, where typical entities include functions, types, and global variables.

3. The *software architecture* level, where typical entities include modules, classes, packages, and subsystems.

We consider "high-level schemas" to be those at the second and third levels of detail.

Reverse engineering tools may support extraction and modelling of system information at any or all of these levels. Typically, fact extractors produce output at level 1 (*e.g.,* Datrix) or level 2 (*e.g.,* Rigi, PBS, Acacia). If higher level views are desired, the reverse engineering tool will provide some kind of abstraction or transformation mechanism. For example, the `grok` relational calculator of PBS allows for the creation of higher level views of system facts by using set of predefined scripts.

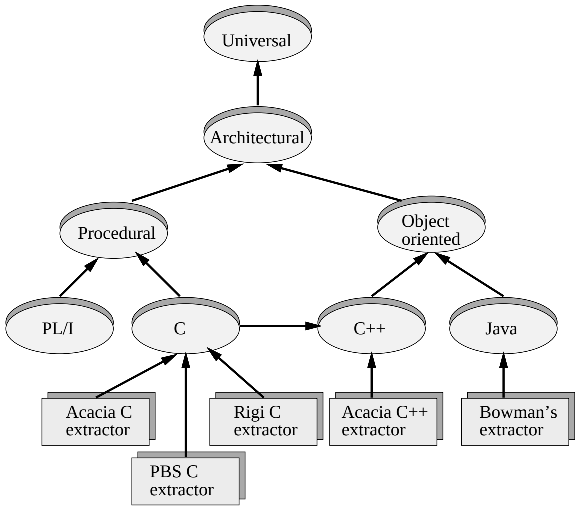

Figure 1. TAXFORM schema hierarchy. An arrow indicates a possible transformation.

3 TAXFORM

The TAXFORM (TA eXchange FORMAT) project [2, 5] has sought to investigate several issues related to the exchange of facts between reverse engineering environments, including:

1. the definition of a set of high-level schemas,
2. automated transformation between different schemas, and
3. the practical problems of adapting different extractors for use within a common environment.

Figure 1 shows the hierarchy of schemas in TAXFORM. The lowest level of detail we consider is the program entity level. We have created several converters for different extractors including Rigi and Acacia (both C and C++) [2, 5, 1]; additionally, there is work in progress adapting the `cxref` and BAUHAS extractors. Example schemas are defined at various levels of detail; for example, Fig. 2 shows the generic procedural language schema and Fig. 3 shows the architectural schema.

The program entity level facts are converted from the native representation of the particular extractor in the TA language by means of customized adapters; the converted facts may then be transformed automatically to other levels of abstraction using the grok tool. We note that our inter-schema transformation scripts are based heavily on those provided by the PBS system.

We consider the main contributions of the TAXFORM project to be twofold: first, the successful adaptation of different extractors into a common framework as well as an

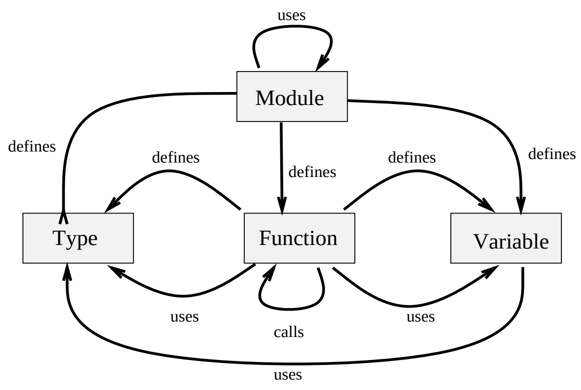

Figure 2. The TAXFORM procedural language schema. Entity attributes are not shown here.

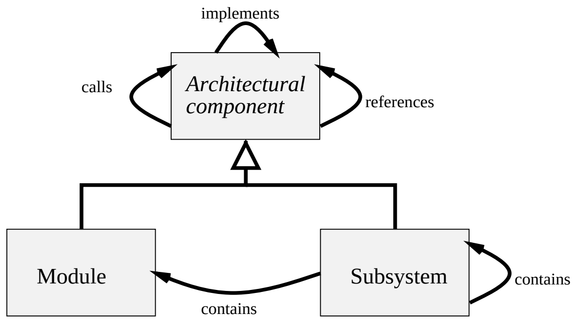

Figure 3. The TAXFORM architectural schema. *Architectural component* **is an abstract entity.**

identification of the common and likely problems in such adaptations, and second, the creation of a set of high-level schemas that may be used with different extractors and even programming languages.

4 WoSEF and High-Level Schemas

Recently at the Workshop on Standard Exchange Format (WoSEF), we summarized and presented the high-level schemas of several different reverse engineering tools [3], including PBS, Acacia, cxref, ctags, TKSee/TA++, BAUHAUS, GuPRO, SHORE, and Neuhold. In addition, developers of other tools such as Rigi, Datrix, MOOSE, and SPOOL also shared their views and experiences. In examining these schemas, we noticed several points of variation:

Level of detail — Different tools extracted different amounts of detail and at different levels of abstraction. For example, the Datrix tool extracts a full ASG, Acacia creates a detailed model of program entities at the external declaration level (including signatures), and PBS extracts relatively little information about program entities apart from name, kind, and containing file.

Intended use and assumptions — The assumptions made by some tools made it difficult to reuse extractors of other tools (or be used by them). For example, the BAUHAUS tool does not extract complete information about file inclusion (it ignores "gratuitous" includes). The TKSee tool, on the other hand, requires modelling of relationships in C code before pre-processing occurs; most extractors do not support this.

Languages modelled — Many tools provided support for procedural and object-oriented languages such as C, C++, and Java, although support for C++ was less common due to the inherent complexity of the language. Several tools (*e.g.,* MOOSE, SHORE) provided both programming language dependent and independent models for storing program facts, although none seriously addressed the issue of integrating multiple extractors.

The GuPRO system was notable for providing support for COBOL and JCL as well as for its modelling of programs that contained embedded code in a second programming language.

5 Summary

A standard exchange format (SEF) for reverse engineering tools must model software systems at different levels of abstraction and provide mechanisms for automated transformation between compatible schemas. To implement such a format, adapters for different fact extractors must be created. However, our work within TAXFORM and discussions at WoSEF suggest that this is non-trivial: some outstanding issues (*e.g.,* agreement on a UID generation convention) must first be addressed, and we observed that some extractors make fundamental assumptions about software that may make them incompatible for use with other tools. We conclude therefore that a more in-depth study of "extractor space" is needed before proceeding with the design of a set of schemas for the SEF.

References

[1] I. T. Bowman, M. W. Godfrey, and R. C. Holt. Extracting source models from Java programs. Available from `http://plg.uwaterloo.ca/~migod/papers/`, 1999.

[2] I. T. Bowman, M. W. Godfrey, and R. C. Holt. Connecting architecture reconstruction frameworks. *Journal of Information and Software Technology*, 42(2), February 2000.

[3] M. W. Godfrey. High-level schemas: A journey through the bush. Presentation given at WoSEF'00, June 2000. Available at `http://plg.uwaterloo.ca/~migod/papers/`.

[4] M. W. Godfrey. Practical data exchange for reverse engineering frameworks: Some requirements, some experience, some headaches. In *Proc. of the ICSE 2000 Workshop on Standard Exchange Format (WoSEF'00)*, Limerick, Ireland, June 2000.

[5] M. W. Godfrey and E. H. S. Lee. Secrets from the monster: Extracting Mozilla's software architecture. In *Proc. of the Second Intl. Symposium on Constructing Software Engineering Tools (CoSET'00)*, Limerick, Ireland, June 2000.

[6] R. C. Holt. An introduction to TA: The Tuple-Attribute language. Available at `http://plg.uwaterloo.ca/~holt/papers/ta.html`, 1997.

[7] R. C. Holt, R. Koshke, and S. E. Sim, editors. *Proc. of the ICSE 2000 Workshop on Standard Exchange Format (WoSEF'00)*, Limerick, Ireland, June 2000.

[8] R. C. Holt, A. Scuhrr, and A. Winter. GXL: Toward a standard exchange format. Technical Report RR-1-2000, University of Koblenz, Koblenz, Germany, 2000.

[9] R. Kazman, S. Woods, and J. Carriere. Requirements for integrating software architecture and reengineering models: CORUM II. In *Proc. of 5th Working Conference on Reverse Engineering (WCRE'98)*, Honolulu, Hawaii, October 1998.

[10] S. Woods, L. O'Brien, T. Lin, K. Gallagher, and A. Quilici. An architecture for interoperable program understanding tools. In *Proc. of the 1998 Intl. Workshop on Program Comprehension (IWPC'98)*, Ischia, Italy, June 1998.