Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Electronic Notes in Theoretical Computer Science 137 (2005) 65–75
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

# Factbase Filtering Issues in an Ontology-Based Reverse Engineering Tool Integration System

Dean Jin[1]

*Department of Computer Science*
*University of Manitoba*
*Winnipeg, Canada*

James R. Cordy[2]

*School of Computing*
*Queen's University*
*Kingston, Canada*

**Abstract**

The *Ontological Adaptive Service-Sharing Integration System (OASIS)* facilitates reverse engineering tool interoperability by sharing services among tools that represent software in a conceptually equivalent manner. OASIS uses a domain ontology to record the representational and service-related concepts each tool offers. Specialized adapters use a filtering process to map factbase instances to domain ontology concepts and apply shared services. This paper examines three issues related to the filtering process: representational correspondence, loss of precision and information dilution.

*Keywords:* reverse engineering, tool integration, interoperability, service-sharing, domain ontology, factbase filtering

## 1 Introduction

Previous approaches to reverse engineering tool integration have concentrated on the exchange of data through specialized hardcoded interfaces (APIs) or

[1] Email: djin@cs.umanitoba.ca
[2] Email: cordy@cs.queensu.ca


doi:10.1016/j.entcs.2005.07.006

rigid standardized exchange formats [13]. These methods have failed to provide software maintainers with a unified environment for supporting and automating reverse engineering tasks [2,6,9,11]. This is because these approaches are *prescriptive*, forcing tool developers to provide a particular functionality to another tool or conform to an idiomatic standard in order to participate in the integration process.

The *Ontological Adaptive Service-Sharing Integration System (OASIS)* is a novel approach to integration that provides a means for reverse engineering tools to work cooperatively to share services and assist maintainers in carrying out reverse engineering tasks. OASIS makes use of specially constructed, external tool adapters and a domain ontology to facilitate integration among a set of reverse engineering tools. A proof of concept implementation of OASIS was recently carried out by researchers involved in the Software Design Ontology Project at Queen's University. This implementation was successful in sharing services among three reverse engineering tools: *ASDT* [8,1], *Fahmy Tool* [4] and *Rigi* [10,12]. This paper provides an overview of OASIS and focuses on three issues that arise from the use of factbase filtering in the integration process.

## 2 OASIS Overview

In an OASIS implementation, a set of reverse engineering tools are selected to participate in an *integration*. Each tool offers a set of services to the integration that are shared among the other participants. A tool service is the functionality provided by a tool that, when given a set of one or more inputs, generates a corresponding output that is relevant for maintainers. In the case of reverse engineering tools the inputs are typically source code (or facts about source code) and the output is typically a report or visualization.

Figure 1 provides an architectural overview of OASIS. The components in an integration consisting of two participant tools ($T_1$ and $T_2$) are shown. An actual OASIS implementation can have any number of participants. Only two are shown here for simplicity. Each tool consists of a factbase *instance* ($I_1$ and $I_2$) containing software facts whose form is dictated by a *schema* ($S_1$ and $S_2$). A set of transactions ($Q_1$ and $Q_2$) conform to the schema and operate on the instance. In [7] we argued that most reverse engineering tools are database systems that are specially tailored to store, manipulate and analyze software facts. The terminology (i.e. transaction, schema, instance) we use to describe the operational characteristics of reverse engineering tools is equivalent to the terms used by researchers in the database systems community (for example, see [3]).

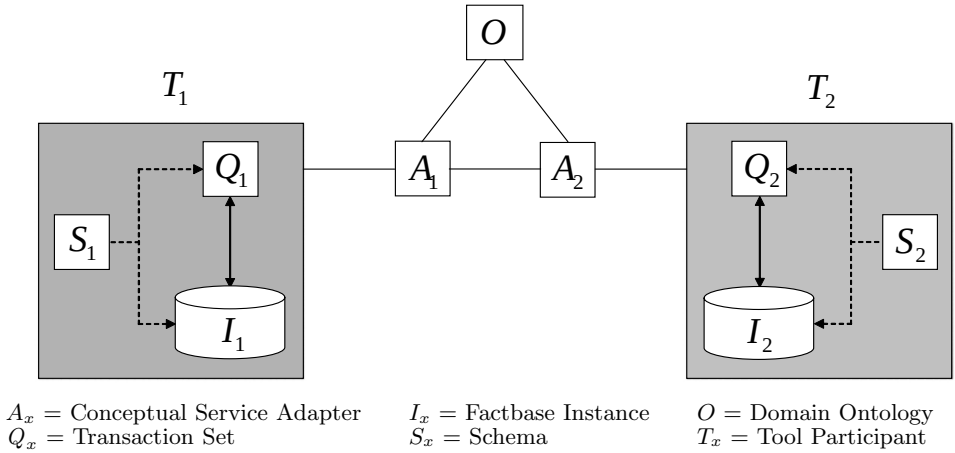



Fig. 1. The OASIS Architecture

An OASIS implementation involves the construction of two types of components: a *domain ontology* ($O$) and tool-specific *conceptual service adapters* ($A_1$ and $A_2$). All the knowledge required to support service-sharing among each of the tools participating in the integration is stored in the domain ontology. Our domain ontology is a tabularized, cross-referenced compilation of shared representational concepts and services offered by each participant in the integration. Only one domain ontology is constructed for an OASIS implementation. The conceptual service adapters (CSAs) operate as integration facilitators. One CSA is affiliated with each integration participant. Although all the CSAs have the same architecture and operational characteristics, each is tailored to handle the functional and information filtering aspects of it's corresponding tool that are required to facilitate interoperability. A service offered by a tool participating in the integration can be shared only when the concepts required by the service intersect with the concepts supported by another participant tool.

## 3 Factbase Filtering

The domain ontology has knowledge of the concepts that are shared among all the tools participating in the integration. Taken together, these concepts define a *conceptual space*, consisting of conceptual 'slots' that factbase instances fit into. A factbase instance fits into a slot when the concept it represents matches a concept in the domain ontology. The representation in the concep-

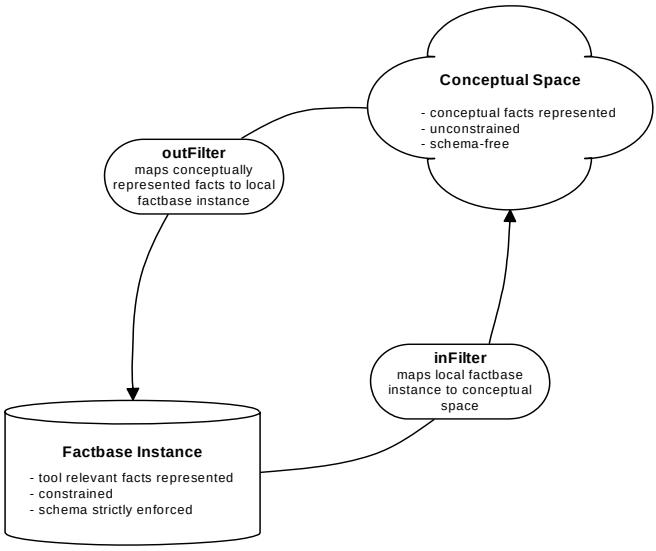



Fig. 2. inFilters, outFilters and the Conceptual Space

*Each conceptual service adapter uses an inFilter and outFilter to respectively map software facts to and from a factbase instance and the conceptual space defined by the domain ontology. Facts stored in a factbase always conform to a tool schema. The conceptual space representation is not bound by a schema, as facts exist there only if they exhibit a conceptual equivalence to a domain ontology concept.*

---

tual space is based on *conceptual equivalence*, so it is not bound by a schema and is free of constraints.

Shared services only operate on fact instances that actually fit into the conceptual slots. When a service is being shared, the CSAs for the two tools involved map all factbase instances into and out of the conceptual space. We call this process of mapping to and from the conceptual space *filtering*. An *inFilter* maps factbase instances to the conceptual space. An *outFilter* maps conceptually represented facts in the conceptual space back to a tool factbase instance. The relationship between a factbase instance, an inFilter, an outFilter and the conceptual space is shown in Figure 2.

The inFilter and outFilter used by each CSA is specially tailored to work with the representation supported by the tool the CSA corresponds to. In the OASIS proof of concept implementation, the filters were constructed using using a combination of LINUX shell scripts and executable *grok* [5] tool scripts.

It is important to understand the actual filtering that is performed by each of the inFilters and outFilters. Although not formally specified, Figures 3 and 4 show the filtering that occurs for ASDT and Fahmy Tool. In each of these

figures, the left column indicates the entities and relationships from the schema for the tool that take part in the filtering process. The right column shows the domain ontology concepts that make up the conceptual space. The arrows between the two columns indicate the mapping the inFilters and outFilters perform. The inFilters map from the left column to the right. The outFilters map from the right column to the left. For example, in Figure 4 the module entity from the Fahmy Tool schema is mapped to the concept SubProgram in the conceptual space. Likewise, in Figure 3 Containment from the conceptual space is mapped to ASDT's contains relationship.

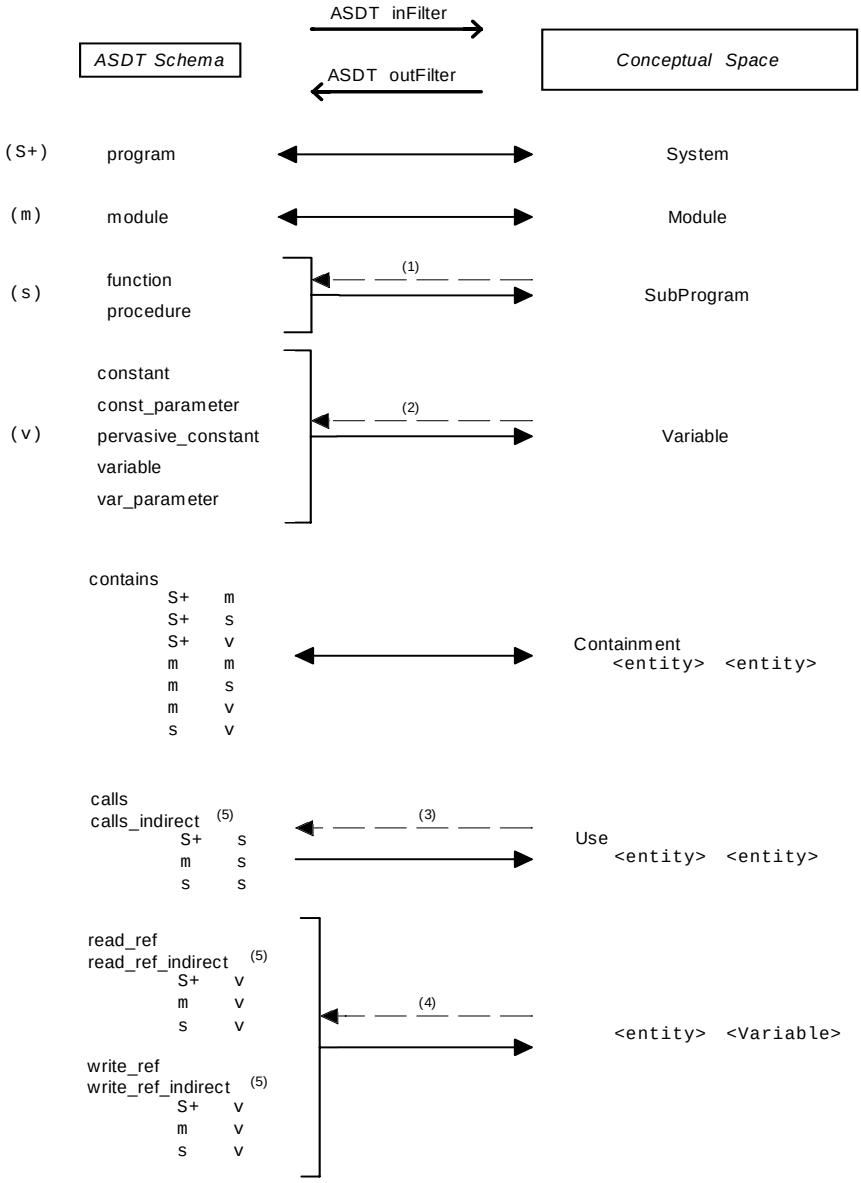
A number of issues related to factbase filtering are apparent in Figures 3 and 4. In particular, we discuss *representational correspondence*, *loss of precision* and *information dilution* in the following sections.

### *3.1 Representational Correspondence*

Representational correspondence is a significant consideration that relates to the mapping of factbase instances to the conceptual space. There are three ways that these mappings can occur:

- **One-To-One**. This mapping occurs when a tool natively represents a concept found in the conceptual space. There is no loss in representational detail in either direction as the factbase instance passes through the conceptual space. For example, 'contains' is commonly represented in the schema for reverse engineering tools and can be considered a one-to-one mapping to the Containment concept in the conceptual space.
- **Fanning Out**. When a single tool schema entry yields more than one conceptual space concept, we refer to the mapping as fanning out. A good example is the usevar relationship in Fahmy Tool (see Figure 4) which corresponds to a Variable entity and a Containment and Use relationship in the conceptual space. We call such a representation 'conceptually rich'. The mapping itself is *constructive*, as it yields three concepts for every usevar factbase instance encountered.

  Note that fanning out is different from *multi-concept equivalence*, where a single tool schema entry is conceptually equivalent to multiple conceptual space concepts. This situation is strongly indicative of concept over classification in the domain ontology. If concept classification is too finely grained, the distinction between concepts becomes blurred and conceptual equivalence for tool schema entries becomes difficult to determine.
- **Fanning In**. In this case, multiple entities or relationships combine together to correspond to a single conceptual space concept. Fanning in is apparent in the ASDT tool (see Figure 3), where the function and procedure




(1) Loss of Precision: SubProgram is reconciled to procedure

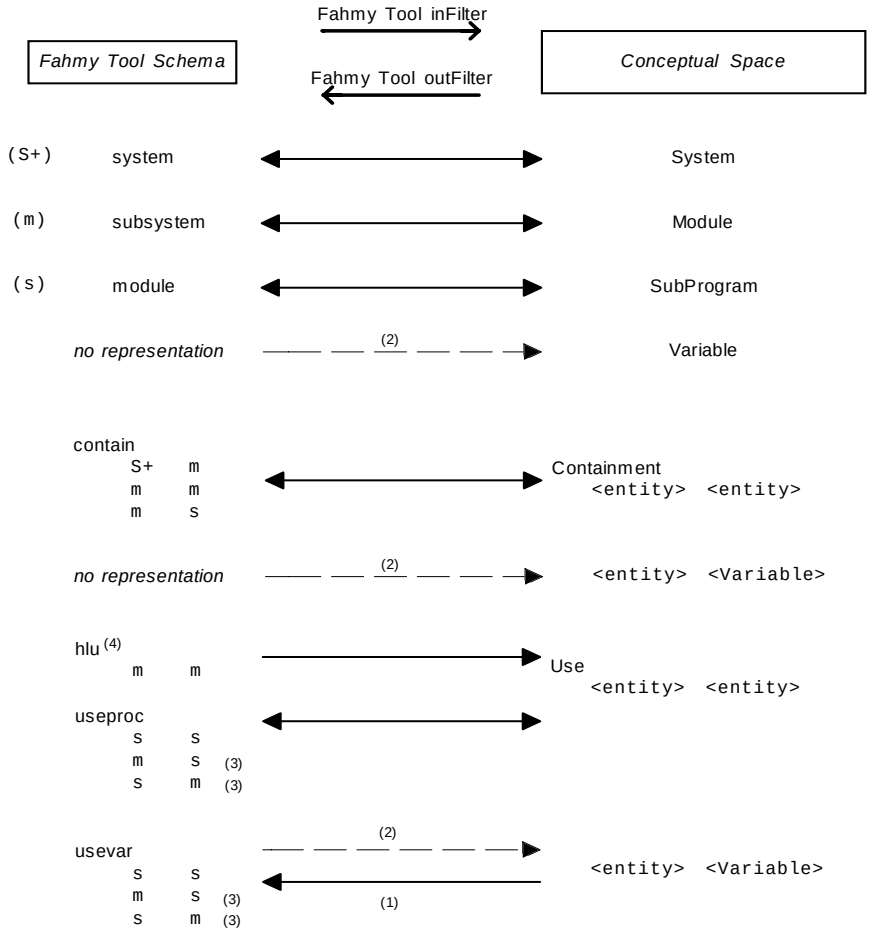
(2) Loss of Precision: Variable is reconciled to variable

(3) Loss of Precision: Use is reconciled to calls

(4) Loss of Precision: Use is reconciled to read_ref

(5) The relationships calls_indirect, read_ref_indirect and write_ref_indirect are produced by the Slice service. The ASDT outFilter does not map conceptually represented facts from the conceptual space into these relationships.

Fig. 3. In and Out Filtering ASDT Schema to the Conceptual Space




(1) This is a special case of Use where each Variable in the range is expressed in terms of the SubProgram that contains it.

(2) Although Fahmy Tool does not support the representation of variables, we are able to induce them from the usevar representation. Unique Variable instances are generated based on the name of the entity where the variable is contained. These instances are identifiable in the range of the usevar relationship. Each usevar instance generates a Variable and Containment and Use relationships in the conceptual space.

(3) These constraints exist in the results of the Hide Interior service. The Fahmy Tool outFilter does not map conceptually represented facts from the conceptual space into these relationship constraints. The inFilter *does* map these fact instances into the conceptual space.

(4) The relationship hlu is produced by the High Level Use service. The Fahmy Tool outFilter does not map conceptually represented facts from the conceptual space into this relationship.

Fig. 4. In and Out Filtering Fahmy Tool Schema to the Conceptual Space

representations together constitute SubProgram in the conceptual space. This mapping is *lossy* because representational detail is lost in the mapping from ASDT to the conceptual space.

### *3.2 Loss of Precision*

*Loss of Precision* is the converse of 'fanning in' representational correspondence discussed in Section 3.1. It occurs when conceptually represented facts correspond to more than one representation in a local tool factbase instance. A single mapping without loss of representational detail is not possible. In this situation the OASIS implementor must decide which local tool representation more closely matches the conceptually represented fact. The outFilter for the tool must be programmed to map those conceptually represented facts to the local representation chosen.

Three examples of loss of precision can be observed in the ASDT outFilter shown in Figure 3. The SubProgram entity is reconciled to procedure, resulting in the loss of the function representation. The Variable entity is reconciled to variable, resulting in the loss of four entity representations: constant, const_parameter, pervasive_constant and var_parameter. Finally, the Use relationship between an <entity> and <Variable> is reconciled to read_ref, resulting in the loss of the write_ref relationship representation.

The negative effects of loss of precision may be minimal. For example, loss of precision for fact instances being brought into a tool for shared-service execution is generally not a problem. The service operates only on the 'lower precision' facts anyway. Nevertheless, results returned from a shared-service may not make sense.

The loss of precision in an OASIS implementation may be an indicator of the need to reevaluate the domain ontology. It may be necessary to expand the ontology to differentiate an important concept from another. This would have the effect of increasing the precision of the ontology so that the distinction between two concepts is no longer lost when it is mapped from the conceptual space.

### *3.3 Information Dilution*

A very important consideration when facilitating shared-services in an OASIS implementation is the problem of *Information Dilution*. It is often the case that the user of a particular tool has expectations of a shared-service that go beyond what the service is capable of providing. Often there are very subtle differences in the representations supported by tools participating in an OASIS implementation. These representational differences affect the fact

instances that get forwarded through the conceptual space to a shared-service and ultimately lead to unexpected results.

Consider the *High Level Use* service offered by Fahmy Tool. In Figure 5 (a) we see the ASDT representation of a hypothetical software system. Executing Fahmy Tool's High Level Use service on this representation yields the results shown in Figure 5 (b). The calls relation between the functions foo and bar is 'lifted' to indicate a high level use relation (labelled hlu) between module 1 and module A.

Now consider a similar ASDT representation shown in Figure 5 (c). Here module 3 calls function foo. Since ASDT supports this representation, we can assume that this representation is not at all uncommon in the software representations supported by the tool. Nevertheless, executing Fahmy Tool's High Level Use service on this representation yields no lifted hlu relation.

In this second situation, the reasons why the results are not as expected are clear. Figure 3 shows that ASDT supports the representation of calls relationships between module and function entities. This representation is preserved when the ASDT inFilter maps them into the conceptual space. Figure 4 shows that Fahmy Tool only supports useproc relationships from module to module entities. The Fahmy Tool outFilter will *not* map into Fahmy Tool all the facts in the conceptual space originally from ASDT. It is at this point that information dilution occurs. As a result, the High Level Use service yields results that the user does not expect.

Information dilution is problematic because it appears that the integration 'works', but the results do not appear correct, even though they are. More troublesome is the possibility that no realization is made that information dilution has occurred. The real problem in the scenario we provide above is that the ASDT user *was not warned* that information dilution had occurred and consequently the results might not be as expected.

## 4 Discussion and Future Work

Perhaps the most difficult aspect to understand about an OASIS implementation is the idea behind the ontology and the conceptual space that it defines. A schema typically defines the allowable characteristics for the information stored in a factbase instance. In OASIS, the domain ontology is a list of representational and service-related *concepts* that tools participating in the integration support. The domain ontology used in an OASIS implementation is not a schema and it does not impose constraints on the conceptual space representation. The goal in OASIS is to identify *conceptual equivalencies* among the representations supported by the tool participants. When two

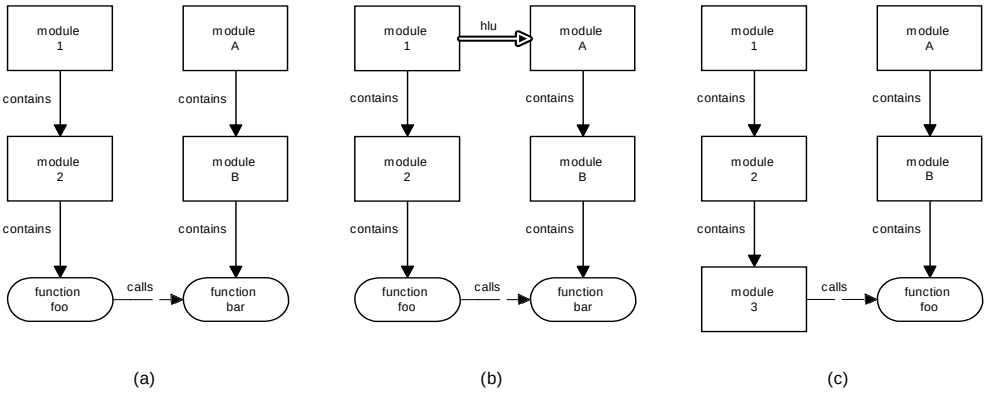



Fig. 5. High Level Use Service (Fahmy Tool)

tools support the same concept, the services that operate on those concepts can be effectively shared.

While the success of the OASIS proof of concept implementation is evident, there is much work that can be done to improve the system. One major consideration is the expansion of the integration to include more tools. In particular, we are interested in exploring the integration of tools that operate at vastly differing levels of abstractive detail with respect to their representations of software. Accommodating these tools would require a significant broadening in the range of concepts that the domain ontology would need to support. Our expectation is that as the ontology grows, so does its usefulness as a facilitator of integration among a wider array of reverse engineering tools. At the same time, augmenting the ontology in this manner would likely introduce a significant amount of detail that would make it much more difficult to manage.

## 5 Conclusion

The OASIS proof of concept implementation successfully demonstrated the feasibility of service-sharing as a means to facilitate the integration of information and services among a set of reverse engineering tools. It has also provided the opportunity to examine issues that arise as a result of the way an OASIS implementation is constructed. In this paper we have examined three areas of concern related to the factbase filtering process used by OASIS to manage the flow of information through the system. Further research is needed to further explore the impact these issues have on reverse engineering tool integration and system integration in general.

# References


[1] Cordy, J. R. and K. A. Schneider, *"Architectural Design Recovery Using Source Transformation"*, in: *CASE'95 Workshop on Software Architecture*, Toronto, 1995.

[2] Ebert, J., B. Kullbach and A. Winter, *"GraX – An Interchange Format for Reengineering Tools"*, in: *Proceedings of the 6th Working Conference on Reverse Engineering (WCRE'99)* (1999), pp. 89–98.

[3] Elmasri, R. and S. B. Navathe, "Fundamentals of Database Systems," Addison-Wesley, 2000, 3rd edition.

[4] Fahmy, H. M., R. C. Holt and J. R. Cordy, *"Wins and Losses of Algebraic Transformations of Software Architectures"*, in: *Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE'2001)*, San Diego, California, 2001.

[5] Holt, R. C., *"Structural Manipulations of Software Architecture Using Tarski Relational Algebra"*, in: *Proceedings of the 5th Working Conference on Reverse Engineering (WCRE'98)*, Honolulu, Hawaii, 1998.

[6] Holt, R. C., A. Winter and A. Schürr, *"GXL: Toward a Standard Exchange Format"*, in: *Proceedings of the 7th Working Conference on Reverse Engineering (WCRE'00) Panel on Reengineering Exchange Formats* (2000).

[7] Jin, D., J. R. Cordy and T. R. Dean, *"Transparent Reverse Engineering Tool Integration Using a Conceptual Transaction Adapter"*, in: *Proceedings of the 7th European Conference on Software Maintenance and Reengineering (CSMR 2003)*, Benevento, Italy, 2003, pp. 399–408.

[8] Lamb, D. A. and K. A. Schneider, *"Formalization of Information Hiding Design Methods"*, in: J. Botsford, A. Ryman, J. Slonim and D. Taylor, editors, *Proceedings of the 1992 Centre for Advanced Studies Conference (CASCON'92)*, Toronto, Ontairo, 1992, pp. 201–214.

[9] Müller, H. A., J. H. Jahnke, D. B. Smith, M.-A. Storey, S. R. Tilley and K. Wong, *"Reverse Engineering: A Roadmap"*, in: A. Finkelstein, editor, *The Future of Software Engineering*, International Conference on Software Engineering (ICSE'00) (2000).

[10] Müller, H. A. and K. Klashinsky, *"Rigi – A system for Programming-in-the-Large"*, in: *Proceedings of the International Conference on Software Engineering (ICSE'88)*, 1988, pp. 80–86.

[11] Perelgut, S., *"The Case for a Single Data Exchange Format"*, in: *Proceedings of the 7th Working Conference on Reverse Engineering (WCRE'00)* (2000).

[12] Rigi Group Home Page, URL: http://www.rigi.csc.uvic.ca/.

[13] Sim, S. E., *"Next Generation Data Interchange: Tool-to-Tool Application Program Interfaces"*, in: *Proceedings of the 7th Working Conference on Reverse Engineering (WCRE'00)* (2000), pp. 278–283.