Available online at www.sciencedirect.com

SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 94 (2004) 81–91

www.elsevier.com/locate/entcs

Issues in Integrating Schemas for Reverse Engineering

Daniel L. Moise[1] and Kenny Wong[2]

Department of Computing Science
University of Alberta
Edmonton, Canada

Abstract

After adopting a standard format such as GXL to exchange graphs of artifacts for reverse engineering tools, the next logical step is to define an appropriate schema for the information contained in the graphs. Various researchers have developed schemas, but in practice, it is still hard to choose an existing one. Typically, researchers end up needing to implement new schemas for the particulars of their tools or case studies. In the paper, we discuss a potential scenario for integrating schemas, with the aim of improving the interoperability among reverse engineering tools.

Keywords: integrating schemas, metaschema, reverse engineering, schema

1 Introduction

Software systems are becoming more and more complex, and the needs for maintaining, understanding, and documenting them are also increasing. Reverse engineering plays an important role in offering solutions to these challenges, and researchers have been conducting case studies to evaluate the capabilities of their reverse engineering tools.

Typically, reverse engineering tools use a fact extractor to obtain information about the source code. These facts have been represented in different formats such as Rigi Standard Format (RSF) [16], Tuple-Attribute language (TA) [8], FAMIX [20], Graph eXchange format (GraX) [4], XML Metadata

[1] Email: `moise@cs.ualberta.ca`
[2] Email: `kenw@cs.ualberta.ca`

doi:10.1016/j.entcs.2004.01.002

Interchange (XMI) [17], or relational databases [19]. It has taken a long time for the reverse engineering research community to adopt a format to exchange the facts among reverse engineering tools. Graph eXchange Language (GXL) [7], the adopted standard exchange format, maintains graphs in eXtensible Markup Language (XML) compliant format [22]. After adopting such a format, the next logical step is to develop metamodels or schemas that offer conceptual modeling of the nodes and arcs for various kinds of graphs. A schema is used to provide information about the types of nodes, arcs, and attributes that are used in the graph, how they are related to each other, and sometimes additional constraints. In most cases, the term "schema" is associated with terms like metamodel, domain model, or conceptual model, depending on the context.

In reverse engineering a software system, one early step involves defining a schema and extracting the artifacts from the source code. These two tasks are mutually dependent and are iterative. As the source code is better understood, more facts are extracted, and the schema evolves to accommodate the new information. Often, an existing schema may not fit as-is to the particular software being analyzed or the tools being used. Consequently, schema reuse is not a simple task, and a proliferation of new schemas has resulted.

Various schemas are used in reverse engineering. Researchers at Bell Canada developed an entity/relation schema, called the Datrix schema, which can represent the abstract syntax tree extracted from C, C++, or Java programs [2,9]. Ferenc and Beszédes proposed a modular schema for C++, called the Columbus Schema, used in the Columbus tool [5]. The Dagstuhl Middle Model (DMM), was developed by a group of researchers that met at the Dagstuhl Seminar on the Interoperability of Reengineering Tools [21]. DMM combines selected ideas from several existing models. We developed a C/C++ schema based on DMM for an industrial case study using the Rigi reverse engineering tool [15].

Some research work has been made to establish a standard schema for C/C++. Ferenc et al. explored building a standard schema for C/C++ at the abstract syntax tree level [6]. Three types of issues were analyzed in the Datrix and Columbus schemas: the lexical, the syntactic, and the semantic structure. The authors concluded that creating a standard schema for C/C++ is a complex problem, which produces many difficulties. Dean and Holt proposed a technique to combine two schemas to create a fact extractor [3].

Despite the availability of existing schemas for C++, when we embarked upon our reverse engineering case study, we could not reuse them for various reasons (e.g., due to limitations in our own tools to import information according to certain schemas, or due to missing elements that were of interest). Thus, we had to develop a new schema for C/C++ to be used with the Rigi reverse engineering tool. Preferably, we would rather spend more time in the analysis of the software system than in writing a fact extractor and its schema.

We discuss the challenge and benefits of integrating existing schemas in Section 2. Section 4 describes in detail an example of integrating two given schemas. Section 3 describes the structuring of schemas in the reverse engineering domain. Section 5 presents our conclusions about this work, and proposes some directions for future research.

2 Integrating Schemas

This section outlines the problems associated with the integration of schemas in reverse engineering.

2.1 Problem

By having easier information exchange and methods to integrate data from diverse, complementary tools, users can form the best tool for the task at hand. Beyond having a common exchange format, it is necessary also to ensure that the exchanged information conforms to common, agreed semantics. The idea of integrating schemas comes naturally, but it is a significant challenge.

There are many schemas in the reverse engineering domain. Since they were developed independently, they have different structures and use different terminology for the entities, relationships, or attributes. Some of these differences identify the same thing, but are expressed in other words. Thus, we need to identify the logically common parts between these schemas. This process is called *schema matching*, and tries to produce a mapping between the elements of the two schemas that match semantically to each other. After this, we can view integrating the two schemas as a union of the two schemas, taking into consideration only once the elements discovered in the schema matching process.

2.2 Challenges

Integrating schemas is a very difficult task. We list some of the issues.

- Different programming languages:
  There are many differences among programming languages, e.g., between Java and COBOL. Each programming language has its own characteristics which bring semantic differences.

- Different levels of abstraction:
  There are a wide variety of facts from statement-level details, high-level structure, and application-domain level constraints.
- Diverse reverse engineering tools:
  Some schemas are tied to modeling limitations with the reverse engineering tool. For example, in Rigi, all node types have the same set of attributes.

2.3 Related Work

The schema integration problem is very old, but it is still not completely solved. It has been investigated in the database field since the early 1980s [1]. Also, this problem is found in the artificial intelligence domain when integrating independently developed ontologies into a single ontology.

Jin et al. proposed a solution for transparent interoperability among reverse engineering tools [11]. They defined a special adapter to deal with a domain ontology, which translates and filters the queries in a conceptual unification, making integration possible. This is similar to creating a common intermediate language for all the reverse engineering tools. The more comprehensive the ontology is, the more chances to obtain a good integration.

Rahm and Bernstein produced a survey of the existing approaches to automatic schema matching [18]. They produced a taxonomy based on different criteria for the matches. They released an algorithm for general schema matching, called Cupid, that includes automated linguistic-based matching [14]. Cupid is based on the elements of the schema, as well as the structure of the schema.

Madhavan et al. proposed a novel approach for matching schemas [13]. The idea of this approach was to extract knowledge from past matching of schemas, and apply this knowledge to match new schemas. Schemas and mappings are added to the mapping knowledge base continually.

3 Schema in Reverse Engineering

3.1 Taxonomy of schemas

Schemas can be categorized according to the *level of abstraction* (which facts are represented in the schema): *low-level* schemas (e.g., Datrix), *middle-level* schemas (e.g., Columbus, CPPDM, and Datrix), and *high-level* schemas.

Two interesting classifications were presented by Jin et al. [10]. For categorizing the schemas, they considered the *definition* of schema (*how* the schema is defined, *implicit* or *explicit*) and the *locality* of schema (*where* the schema is defined, *internal* or *external*).

3.2 Classifying schemas

However, to understand the overlaps, redundancies, and purposes of diverse kinds of schemas, we need some organizing structure to help classify the schemas. One idea is to classify schemas into two main categories: those focused on the programming domain and those focused on the application domain. Each of these categories could be further decomposed. For example, the programming domain could be divided into procedural, object-oriented, markup languages, etc. Similar to the programming domain, the application domain could be decomposed into applications from accounting, Customer Relationship Management (CRM), Supply Chain Management (SCM), and so on. See Figure 1.

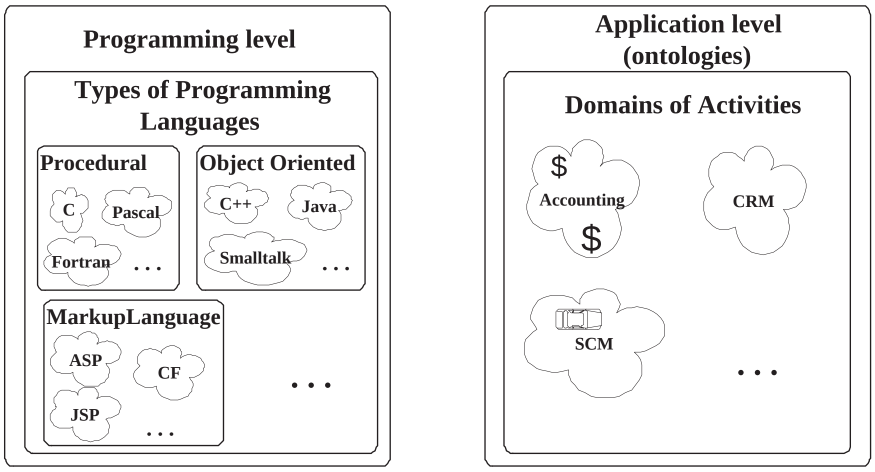


Fig. 1. Classifying schemas

The subcategories contain schemas or integrated schemas as appropriate. For example, in the object-oriented subcategory, the schemas that reside here depend on what arises in practice from the existing schemas for Java, C++, etc., and the tools that deal with these languages.

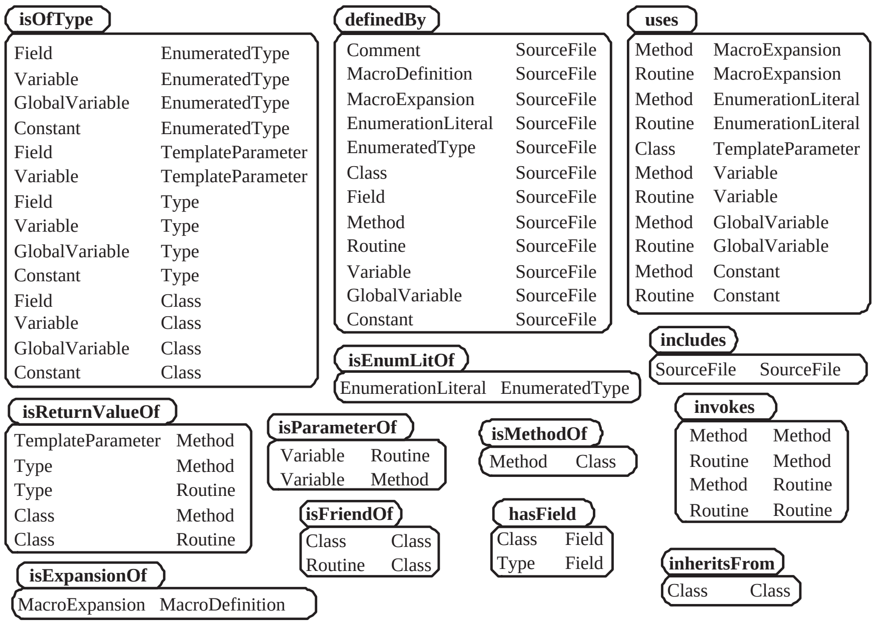
Also, one interpretation of the organizing structure is that the (sub)categories are metaschemas for what resides directly within them. For example, the object-oriented category may be considered a metaschema from which a specific object-oriented language schema could be "instantiated". In a sense, the organizing structure forms a metaschema hierarchy.

4 Example of integrating two schemas

This section presents an example of integrating two schemas, both intended for C/C++ elements. One of them is CPPDM developed by the University of Alberta, and the other one is DMM developed by the University of Ottawa. First, we introduce the two schemas and then we present their integration.

4.1 Rigi C/C++ Domain Model (CPPDM)

Rigi C/C++ Domain Model, or CPPDM, was developed to accomplish a case study in reverse engineering [15]. This domain model is inspired by some elements from DMM. In general, CPPDM is suited for structural facts about any software system implemented in C/C++. The entities and relationships of CPPDM are illustrated in Figure 2.



isOfType

Field	EnumeratedType
Variable	EnumeratedType
GlobalVariable	EnumeratedType
Constant	EnumeratedType
Field	TemplateParameter
Variable	TemplateParameter
Field	Type
Variable	Type
GlobalVariable	Type
Constant	Type
Field	Class
Variable	Class
GlobalVariable	Class
Constant	Class

definedBy

Comment	SourceFile
MacroDefinition	SourceFile
MacroExpansion	SourceFile
EnumerationLiteral	SourceFile
EnumeratedType	SourceFile
Class	SourceFile
Field	SourceFile
Method	SourceFile
Routine	SourceFile
Variable	SourceFile
GlobalVariable	SourceFile
Constant	SourceFile

uses

Method	MacroExpansion
Routine	MacroExpansion
Method	EnumerationLiteral
Routine	EnumerationLiteral
Class	TemplateParameter
Method	Variable
Routine	Variable
Method	GlobalVariable
Routine	GlobalVariable
Method	Constant
Routine	Constant

includes

SourceFile	SourceFile

isEnumLitOf

EnumerationLiteral	EnumeratedType

isReturnValueOf

TemplateParameter	Method
Type	Method
Type	Routine
Class	Method
Class	Routine

isParameterOf

Variable	Routine
Variable	Method

isMethodOf

Method	Class

invokes

Method	Method
Routine	Method
Method	Routine
Routine	Routine

isFriendOf

Class	Class
Routine	Class

hasField

Class	Field
Type	Field

inheritsFrom

Class	Class

isExpansionOf

MacroExpansion	MacroDefinition

Fig. 2. Relationship types

This domain model incorporates fifteen node types and fifty-eight possible arc types between these node types. The node types from CPPDM are: *SourceFile, Comment, MacroDefinition, MacroExpansion, GlobalVariable, Constant, Variable, EnumeratedType, EnumerationLiteral, Type, Class, Method, Field, Routine* and *TemplateParameter*.

The set of attributes for each entity in CPPDM are shown in Table 1.

Table 1
Possible attributes for nodes

Node Type	Possible Attributes
SourceFile	file, path
Comment	file, lineno, position, endlineno, endposition
MacroDefinition	
MacroExpansion	file, lineno, position
EnumerationLiteral	file, lineno, position
EnumeratedType	file, lineno, position
TemplateParameter	
Type	
Class	file, lineno, position, template
Field	file, lineno, position, access, real_type
Method	file, lineno, position, access, binding, real_type, real_type_arg
Routine	file, lineno, position, real_type, real_type_arg
Variable	file, lineno, position, real_type
GlobalVariable	file, lineno, position, real_type
Constant	file, lineno, position, real_type

4.2 Dagstuhl Middle Model (DMM)

Figure 3 illustrates DMM version 0.006 [12]. DMM is intended to represent facts for procedural programs, as well as object-oriented software.

4.3 Integrating CPPDM and DMM

We present a set of steps for integrating two schemas, using CPPDM and DMM as examples. As input, we have two schemas, and as output, we want an integrated schema, together with associated rules to be used for data integration. In essence we create equivalence relations over the entities, relationships, and attributes occurring in the two schemas, and form a new schema based on the identified equivalence classes.

- **Step 1: Mapping of the entities**
  This step finds the common entities between the two schemas. The mapping can be done based on syntax, synonyms, hyponyms, abbreviations, or as suggested by the user (especially if differing semantics are involved).

  We introduce the following notation: E1 $\equiv$ E2 if the entity E1 is mapped to the entity E2. The $\equiv$ operation is commutative and associative. Based on the $\equiv$ operator, an equivalence relation over the entities is formed. In our mapping, we obtain the following equivalences between the entities from CPPDM and that from DMM (on the left side of the $\equiv$ operator is the entity from CPPDM, and on the right side is the entity from DMM):
  SourceFile$\equiv$*SourceFile*, *Comment*$\equiv$*Comment*, *GlobalVariable*$\equiv$*Variable*, *MacroDefinition*$\equiv$*MacroDefinition*, *MacroExpansion*$\equiv$*MacroExpansion*, *Constant*$\equiv$*Variable*, *Variable*$\equiv$*Variable*, *EnumeratedType*$\equiv$*EnumeratedType*, *EnumeratedLiteral*$\equiv$*EnumeratedLiteral*, *Type*$\equiv$*Type*, *Routine*$\equiv$*Routine*,

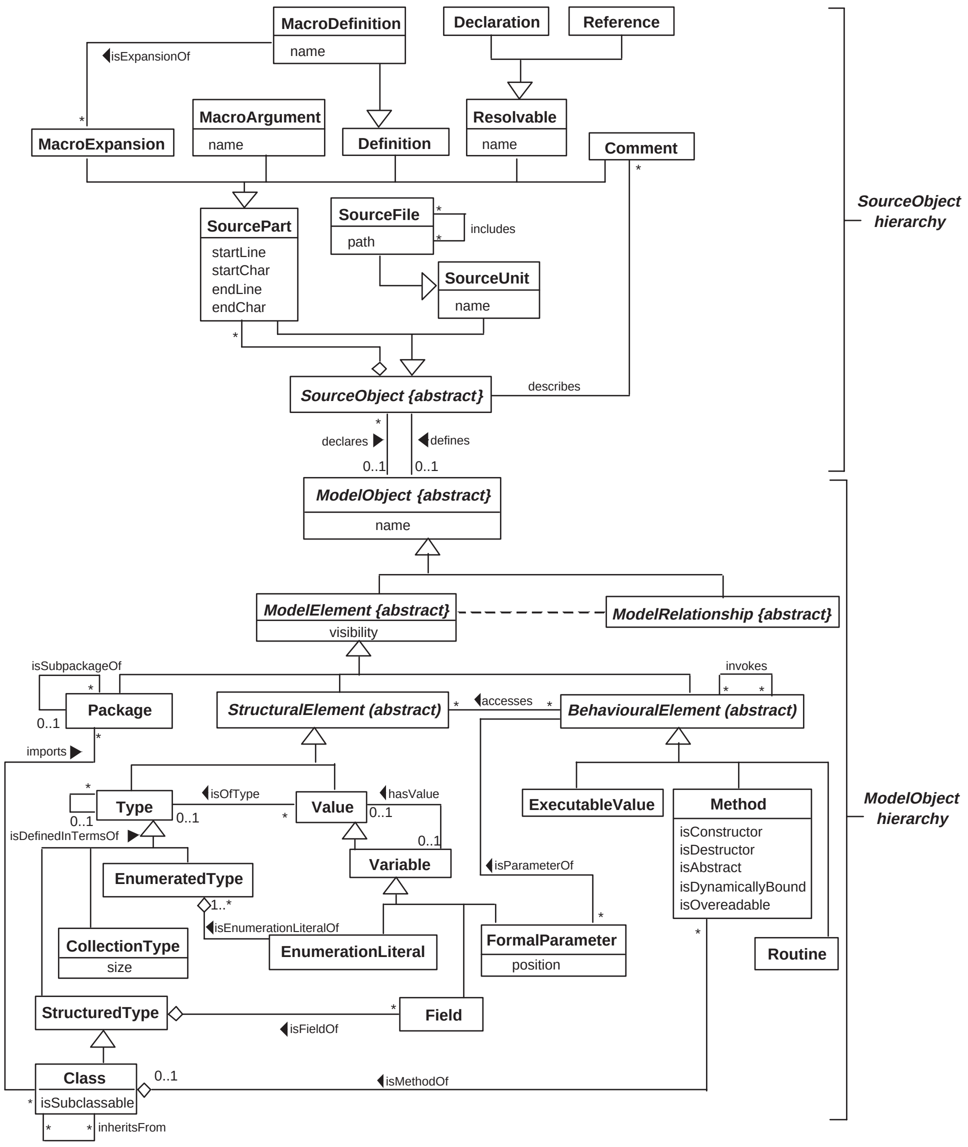


Fig. 3. Dagstuhl Middle Model version 0.006

Method≡*Method*, *Field*≡*Field*, *Class*≡*Class*. Based on the associative property, we obtain the following equivalence class: {*GlobalVariable*, *Constant*, *Variable*}.

- **Step 2: Build the entities**
  Given the mapping between the entities of the schemas, we label the equivalence classes to obtain the entities for the resulting schema.

After this step is applied for CPPDM and DMM, we obtain the following entities for the new schema: *SourceFile*, *Comment*, *MacroDefinition*, *MacroExpansion*, *MacroArgument*, *Declaration*, *Reference*, *Package*, *Type*, *EnumerationLiteral*, *EnumeratedType*, *Variable*, *Class*, *Field*, *Method*, *Routine*, *FormalValue*, *ExecutableValue* and *TemplateParameter*.

- **Step 3: Build the rules for the entities**
The aim of creating an integrated schema is to use it for integrating the data from different fact extractors. For example, if we have a software system that contains elements of C and Tcl, we might extract the facts separately for each language using the appropriate extractor, and store in two different repositories. Each repository conforms to its schema, for C and Tcl respectively. For data integration, we have to maintain the rules for mapping the elements from the initial schemas to the elements of the resulting schema. These rules may involve filtering data from each repository, or transforming the data as needed to conform to the new schema.

  This step builds these rules for the identified equivalence classes over the entities. For example, for the equivalence class {*GlobalVariable*, *Constant*, *Variable*} labeled with *Variable*, we build two rules that unify the *GlobalVariable* and *Constant* entities to the new equivalence class.

- **Step 4: Map and build the relationships**
We build the relationships iteratively between every pair of entities from the new schema. For the relationships, if any, between entities E1 and E2, use a mapping to obtain a set of equivalence classes over the relationships. If all of the relationships in the same equivalence class have the same direction the final equivalence class will take the same direction. Otherwise, the direction of the equivalence class will have some user-specified direction. Note, that we have to add all of the rules for mapping the relationships, including that regarding the direction. Label the equivalence classes, and add them as the new relationships between E1 and E2.

- **Step 5: Map and build the attributes**
For completing the new schema, we have to map and build the attributes based on the attributes in the initial schemas for both entities and relationships. The process of mapping is similar to that for the entities and relationships themselves.

  We provide an example of building attributes for the *Class* entity in the new schema. The attributes for the CPPDM *Class* entity are *file*, *lineno*, *position* and *template*. For the same entity in DMM the single attribute is *isSubclassable*. The set of the attributes for the *Class* entity in the new schema is then: *file*, *lineno*, *position*, *template*, and *isSubclassable*.

5 Conclusions

In this paper, we have outlined the need for integrating schemas for reverse engineering, and illustrated a process for integrating two given schemas. As future work, additional case studies are needed to apply the process for integrating schemas in reverse engineering. Also, we need to refine the derived transformation rules and their relationship to the data integration process. Another need is an approach for further changes of the integrated schema to enhance its reuse for a particular reverse engineering purpose.

References

[1] Batini, C., M. Lenzerini and S. B. Navathe, *A comparative analysis of methodologies for database schema integration*, ACM Computing Surveys **18** (1986), pp. 323–364.

[2] Bell Canada Inc, *Datrix* (2003).
URL http://www.iro.umontreal.ca/labs/gelo/datrix

[3] Dean, T. R., A. J. Malton and R. Holt, *Union schemas as a basis for a C++ extractor*, in: *Working Conference on Reverse Engineering—WCRE 2001 (Stuttgart, Germany)*, 2001, pp. 59–67.

[4] Ebert, J., B. Kullbach and A. Winter, *GraX—an interchange format for reengineering tools*, Technical Report 5–99, Universität Koblenz-Landau, Institut für Informatik, Koblenz, Germany (1999).
URL http://www.uni-koblenz.de/~ist/retrieve/RR-5-99.pdf

[5] Ferenc, R. and A. Beszédes, *Data exchange with the Columbus schema for C++*, in: *Sixth European Conference on Software Maintenance and Reengineering—CSMR 2002 (Budapest, Hungary)* (2002), pp. 59–66.

[6] Ferenc, R., S. E. Sim, R. C. Holt, R. Koschke and T. Gyimóthy, *Towards a standard schema for C/C++*, in: *Working Conference on Reverse Engineering—WCRE 2001 (Stuttgart, Germany)*, 2001, pp. 49–58.

[7] GUPRO, *Graph eXchange Language (GXL)* (2003).
URL http://www.gupro.de/GXL

[8] Holt, R. C., *An introduction to TA: The Tuple-Attribute language.* (2002).
URL http://plg.uwaterloo.ca/~holt/papers/ta-intro.htm

[9] Holt, R. C., A. E. Hassan, B. Lague, S. Lapierre and C. Leduc, *E/R schema for the Datrix C/C++/Java exchange format*, in: *Working Conference on Reverse Engineering—WCRE 2000 (Brisbane, Australia)*, 2000, pp. 284–286.

[10] Jin, D., J. R. Cordy and T. R. Dean, *Where's the schema? A taxonomy of patterns for software exchange*, in: *Tenth International Workshop on Program Comprehension—IWPC 2002 (Paris, France)* (2002), pp. 65–74.

[11] Jin, D., J. R. Cordy and T. R. Dean, *Transparent reverse engineering tool integration using a conceptual transaction adapter*, in: *Seventh European Conference on Software Maintenance and Reengineering—CSMR 2003 (Benevento, Italy)*, 2003, pp. 399–408.

[12] Lethbridge, T. C., *The Dagstuhl middle model: An overview*, in: *First International Workshop on Meta-models and Schemas for Reverse Engineering—ateM 2003 (Victoria, BC, Canada)*, 2003.

[13] Madhavan, J., P. A. Bernstein, K. Chen, A. Halevy and P. Shenoy, *Corpus-based Schema Matching*, in: *Eighteenth International Joint Conference on Artificial Intelligence—IJCAI 2003 (Acapulco, Mexico)*, 2003.

[14] Madhavan, J., P. A. Bernstein and E. Rahm, *Generic schema matching with Cupid*, in: *International Conference on Very Large Data Bases*, 2001, pp. 49–58.

[15] Moise, D. L. and K. Wong, *An industrial experience in reverse engineering*, in: *Tenth Working Conference on Reverse Engineering—WCRE 2003*, 2003, pp. 275–284.

[16] Müller, H. A., M. A. Orgun, S. R. Tilley and J. S. Uhl, *A reverse engineering approach to subsystem structure identification*, Journal of Software Maintenance: Research and Practice **5** (1993), pp. 181–204.

[17] Object Management Group (OMG), *XML Metadata Interchange (XMI)* (2003).
URL http://www.omg.org/technology/documents/formal/xmi.htm

[18] Rahm, E. and P. A. Bernstein, *A survey of approaches to automatic schema matching*, International Journal on Very Large Data Bases **10** (2001), pp. 334–350.

[19] Sneed, H. M. and T. Dombovari, *Comprehending a complex, distributed, object-oriented software system a report from the field*, in: *International Workshop on Program Comprehension (IWPC)* (1999), pp. 218–225.

[20] University of Berne, *Famix* (2003).
URL http://www.iam.unibe.ch/~scg/Archive/famoos/FAMIX

[21] University of Ottawa, *Dagshtuhl Middle Model (DMM)* (2003).
URL http://scgwiki.iam.unibe.ch:8080/Exchange/2

[22] World Wide Web (W3C), *Extensible Markup Language (XML)* (2003).
URL http://www.w3.org/XML