Interoperability of Reverse Engineering Tools

21.01. - 26.01.2001

organized by

Jürgen Ebert – University of Koblenz, Germany
Kostas Kontogiannis – University of Waterloo, Canada
John Mylopoulus – University of Toronto, Canada

Software Reengineering is the present-day term for all activities for renovating aging systems to be more responsive to changes. Problems of the 90s like the Y2k-problem or the problem of converting software to the new European currency witnessed the importance of concepts, tools and techniques to improve the quality and maintainability of software.
Reengineering is a part of software engineering with its focus on all problems appearing during software maintenance of legacy software. In this seminar we focused on the technical part related to the software artifacts themselves and excluded the management aspects of software maintenance.
Reengineering activities use to a large part the same concepts, tools and techniques as other software engineering disciplines. Besides software engineering knowledge, there is also much usage of other more traditional areas of computer science, especially compiler construction, database systems, formal semantics, and knowledge representation. But due to its special focus, additional problems appear, like

- reverse engineering
  (recognition of architecture, cliches, procedures, structure, and redocumentation),
- migration
  (change of database models and/or programming languages), and
- program understanding
  (querying, browsing, visualization techniques, concept analysis).

Software Reengineering activities are widespread and mostly focus on the development of tools for software analysis. Thanks to several conferences and workshops in this area such as CSMR, ICSM, IWPC and WCRE a common terminology and definitions of agreed aims of reengineering research are slowly emanating.

The current state of practice is that reengineering tools still solve insular problems and are treated as research prototypes within the research group that developed them. In this context, it is very important to define a data interchange format that allows for different reengineering tools to communicate so that integrated, multi-faceted representations of software systems can be created.
Even though this issue may look simple at the beginning, it involves a number of research issues to be resolved. One issue is the definition of the levels of abstraction that information about a software system is to be presented on. These levels of abstraction may include the abstract syntax tree level, the data and control flow level, or the architectural level.
The challenge to the research community is to design a formalism for each level of abstraction so that information about a software system can be passed from one analysis tool to another. Moreover, the formalism must allow for software systems and constructs in various languages to be presented. Another research challenge is the definition of schemas that allow for data emitted from different parsers to be fused in a uniform, normalized source code representation. Emerging markup languages such as XML may provide a vehicle for data fusion and data integration in this context.
The actions done for enhancing interoperability of research and tools in reengineering have only partially been successful up to now. Schemas for a number of popular languages such as C, Cobol, Fortran and, PL/I have been developed. Languages and supporting tools that allow for architectural descriptions to be specified and exchanged in the form of tuples have been implemented by various groups.

At this 5-day Dagstuhl seminar 47 people gathered together in order to discuss, extend and combine the work done by various groups into a common confluent and coherent result. This report collects the abstracts of all talks given during this week covering all relevant aspects of interoperability - including metamodels, concrete tools and frameworks, practical experiences and relevant technologies like XML. One afternoon was used for a demo session where several tools were introduced to the participants (db-main, shrimp, codecrawler, gupro, columbus, ta, missinglink, fujaba).
The participants agreed on the necessity of interchange languages on different levels of semantic expressiveness, like

- abstract syntax trees and abstract syntax graphs,
- call graphs and program dependence graphs,
- architecture descriptions.

XML was regarded as a vehicle for reengineering tool interoperability. GXL, a proposal for an XML-based basic format for interchanging graph-like reengineering data, had been developed during the last months before the seminar by the

cooperation of several groups, especially Waterloo (Ric Holt), Toronto (Susan Elliott Sim), Koblenz (Andreas Winter) and Munich (Andy Schürr). GXL is an XML-based amalgamation of existing formats. On friday, January 26th, the participants of this seminar accepted GXL 1.0 (meanwhile known as the Dagstuhl-version) as an interchange language and more than twenty participants assured that they were going to make their tools interoperable on the basis of GXL. The further development of GXL can be inspected at `http://www.gupro.de/GXL/`.
Beyond a common format, tool interoperability highly relies on the agreement of common concepts, schemas, and information structures. During the workshop the participants discussed such schemas in four groups focussing on different aspects: syntax level, middle level, architecture level and data level. The discussions started by these groups did not yet lead to mature proposals, but this work is still being continued. Concerning the C++ syntactical schema a lively email-list (`gxl-cpp@rgai.inf.u-szeged.hu`) was founded where the discussion is continued. Concerning the middle level a wiki as been installed at `http://scgwiki.iam.unibe.ch:8080/Exchange/2`.
So this seminar was not only a productive and pleasant week for all participants, largely thanks to the well thought-out setup and the quality of service that Dagstuhl provides. It also was the source of continuing work on interoperability of reengineering tools.
We thank the Dagstuhl staff for their competent, efficient and friendly support for this productive and enjoyable week.

The Organizers

Jürgen Ebert — *Kostas Kontogiannis* — *John Mylopoulos*

Powerpoint slides and/or PDF files of most talks can be found on the Schloß Dagstuhl Website of Seminar 01041.

Contents

1 Interoperability of Reengineering Tools: Two Years Back, Two Years Ahead

Rainer Koschke, University of Stuttgart, Germany
Susan Elliott Sim, University of Toronto, Canada

In this talk, we presented a tool interoperability maturity model and discuss the past and necessary future efforts to achieve higher levels on the model. The model has three levels: 0) ad hoc; 1) static interoperability through an exchange format; and 2) dynamic interoperability through a control infrastructure. In 1998, there were two groups (one Canadian and one international) working to develop a standard exchange format. By 2000, these two grops joined their efforts and through a series of meetings and workshops achieved a high leve of consensus, where almost two dozen groups have agreed to work towards making GXL the standard exchange format. It is expected that version 1.0 of GXL be compelted and ratified by the end of this seminar. However, to acheive level 1 some reference will also need to be developed, specifically ones for the abstract syntax tree level, the program entity level, and the architectural level. This is work is also anticipated during this seminar along with other discussions to achieve level 2. The success of the interoperability of tools depends significantly on the success of interoperability of people through consultation and consensus-building. The achievements over the last two years have been based on these principles and we anticipate continuing to rely on them in making progress in the future.

2 Models for Reverse Engineered Artifacts

Timothy C. Lethbridge, University of Ottawa, Canada

This presentation discusses how to represent information about software that extracted during the reverse engineering process, and either stored in a database or transmitted among tools. Other talks have discussed the need for a syntax for such information, the general consensus bing that GXL is a good candidate. This talk focuses on the *schema* to be used. The talk first considers the premise that the UML metamodel might be a good candidate. However the UML metamodel represents high-level design information, ignoring the fact that there is

a need to represent the structure of the source code. For example, the UML metamodel does not handle such things as file inclusion, and the precise position in source code of such things as definitions, declarations and references. In the talk, we then discuss a UML model whose top-level class is called SourceElement. Important lower level sub-hierarchies include SourceUnit (entire editable source elements such as files), SourcePart (parts of SourceUnits, including Definitions and Resolvables, which can be Declarations or References), SourcePackages (Collections of source units) and ReferenceExistences. The latter class represents the relationships between source code entities (e.g. calls, inclusions etc.) without the full detail of Resolvables.

3 Graph based Reverse Engineering and Reengineering Tools

Katja Cremer, RWTH Aachen, Department of Computer Science III, Germany (now at Ericsson Eurolab)

In this talk the data formats of a graph based reverse and reengineering tool are presented. The tool has been developed in a project with two industrial partner. The main scope of the projekt is the preparation of existing COBOL programs for the tense in distributed environments.
The first step is a design recovery to get a higher level of abstraction. The information on this level are presented as directed graphs. The tool offers transformations to restructure the existing programs. The transformations are performed on the abstract graph level. These transformations are connected with source code transformations to adapt the concrete underlynng source code according to redisign steps.
The existence of a exchange format would simplify the development of graph based reverse and reengineering tools.
GXL seems to be a good approach. But first some use cases would be helpful.

4 Moose - Interoperability in XMI

Sander Tichelaar, University of Berne, Switzerland

The talk introduces Moose - our language-independent environment for reengineering object-oriented systems - in the context of tool interoperability. It introduces the metamodel (or schema) it uses to store and exchange program-entity level information. Some of the design decisions such as naming schemes and language mappings are discussed, and the fact that we do not use UML because of its focus on OOAD rather than representing source code. The actual exchange of information is done using XMI. Advantages of this approach is that it is a standard and that it is model-based: DTDs and XML files can be generated automatically and the common metametamodel (MOF) can be used to integrate with other models. Disadvantages of XMI are its verbosity and the way it deals with model extension. Both RDF and CDIF are more flexible in this last respect. Finally a few points are mentioned that any model should deal with and that not seem to be covered by the current approaches such as unique naming, incremental loading of information and support for multiple models.

5 What can we do while waiting for a common exchange format for reverse engineering tools?

Spiros Mancoridis, Drexel University, Philadelphia, USA

(we might have to wait for a long time ...)
Together, research groups at Drexel University and the AT&T Labs - Research have developed reverse engineering tools to support:

- static source code analysis for C/C++ and Java (the Acacia and Chava tools)
- dynamic analysis for Java executable code (the Form library and the Gadget tool)
- software clustering (the Bunch tool)
- architecture-level dependency induction (the ISF tool)

Of course, there are many other reverse engineering tools developed elsewhere. Lately, the reverse engineering community has been discussing exchange formats as a method to integrate tools for reverse engineering. This is a noble goal that unfortunately, if history is an indicator, may never be realized. Many may remember the tool integration goals set by CASE tool vendors and researchers in the early 90's. Although several exchange formats and tool integration standards were proposed (e.g., CDIF, PCTE), these efforts were largely ignored by both researchers and CASE tool vendors.
In the meantime, we are taking a different approach to tool integration as we wait to see what will emerge from the community regarding tool integration and data exchange. Specifically we are developing an on-line Reverse Engineering Portal site, called REportal, which will enable programmers to upload their code to a secure server running at Drexel University and use wizards to guide them through the various reverse engineering services that will be provided by the portal. The portal will present services (not tools) to the users and the tool integration will be done behind the scenes via scripts. The portal simplifies the reverse engineering process in two ways: 1) users will not have to learn the specifics of any tool, as the wizards will provide a high-level abstraction to the portal services 2) users will not need to install any tools on their machines; thus protecting them from mundane, yet time consuming, problems such as keeping track of bug fixes, new releases, incompatible operating systems and libraries, different file formats, and so on.
This work is a collaborative effort involving Drexel University's Spiros Mancoridis and Timothy Souder as well as AT&T Lab's Yih-Farn Chen, Emden Gansner, and Jeff Korn

6 Re-Engineering Tools for Multi-Language Multi-Pargadigm Software

Panagiotis K. Linos, Butler University, Indianapolis, USA

It has been documented that more than 30% of software applications today are written in more than two programming languages. In this talk, I presented various issues related to the construction and interoperability of re-engineering tools for multi-language multi-paradigm (MM) software. Specifically, a research framework was presented for formalizing, managing, storing and experimenting with MM program dependencies, found in MM software.

7 A general meta-model for data-centered application reengineering

Jean-Luc Hainaut and Jean Henrard, University of Namur, Belgium

Reengineering projects have strong requirements as far as models and engineering processes are concerned. Three examples: (1) data and processing descriptions are more tightly linked than in forward engineering, (2) data models must describe many features that generally are out of scope of standard forward engineering models (for instance, the UML class model is quite inadequate for data reverse engineering processes), (3) in reverse engineering, the very concept of "abstraction level hierarchy" is blurred due to the coexistence, in the same schema, of constructs pertaining to different abstraction levels. The presentation will first state the main requirements of the reengineering processes. Then, a meta-model that tries to meet them will be described. Finally, we will show how this meta-model has been implemented into DB-MAIN, a wide scope reengineering CASE environment.

8 GXL - Graph eXchange Language

Ric Holt, University of Toronto, Canada
Andy Schürr, University BW Munich, Germany
Susan Elliot Sim, University of Toronto, Canada
Andreas Winter, University of Konblenz, Germany

The fields of reverse engineering and program analysis have matured to the extent that there are many tools to extract information about programs, to manipulate this information and to analyze it. What is missing is a generally accepted means to allow these tools to interoperate.
GXL offers a graph-based format supporting tool interoperability on the data interchange level. GXL has evolved from many discussions among groups developing reengineering tools. It was also influenced by similar activies on tool interoperability in graph drawing and graph transformation.
In GXL instance data and their according schema data is represented by typed, attributed, directed graphs. Both, instance data and schema data are exchanged as XML documents following ONE common DTD (document type definition).

The repesentation of graphs and schemas follows the notation given for UML object and class diagrams.
The talk introduces into the foundations of GXL, presents the XML-based exchange language for graphs, shows the graphical notation used to represent graphs and schemas to a human reader, and defines the GXL meta schema.

9 Tool Interoperation in Large-Scale Euro Transition Projects

Rainer Gimnich, IBM Global Services, Stuttgart, Germany

The introduction of the single European currency (the EURO) entails a wide-ranging adaptive software maintenance problem to the companies and public administrations in the 'Euroland' countries.
In the talk we present a methodology and a set of tools to support practical Euro reengineering projects. The tools range from analysis tools to database migration tools. Each tool has a different origin and, for project purposes, their functionalities have been extended, and interfaces have been built to enable tool interoperation.
With this approach, large scale (up to 100 MLoc) projects can be performed and managed efficiently.

10 Reverse Engineering Experiences at Nokia

Claudio Riva, Nokia Research Center, Finland

We present a reverse engineering experience that has been carried out in Nokia during the European project FAMOOS. The experience aimed at (1) assessing the quality of a Nokia software system using the tools developed in the project and (2) evaluating the FAMOOS reverse engineering tools in practice with an industrial case study.
The experience has been organised in two steps. In the first phase, the nine developers of the tools have worked together for four days to detect shortcomings

in the system and propose solutions. In the second phase, they have presented the results to the programmers of the system in a panel. The reverse engineering team worked as a single taskforce, in fact they had daily meetings to exchange findings, suggest new ideas and plan the work ahead.
An interesting problem has been detected in the design of the core classes of the system and a solution has been proposed. The developers of the system confirmed the presence of that problem in the system and welcomed the proposed solution. The experience shows that in few days a group of nine people managed to understand a lot of an unknown system using the reverse engineering tools. The experience also shows that reengineering is a team work where team-communication plays an important role and different reverse engineering approaches are necessary.

11 Architecture Analysis Challenges in the Medical Imaging Domain - Why we need interoperability of reengineering tools

Tobias Rötschke, Philips Research Laboratories Eindhoven, The Netherlands

Medical imaging systems are rather expensive SW intensive embedded systems, and hence maintained over a long lifetime of typically 10 to 20 years. Different modalities (based on different imaging technologies) differ widely from the architecture analyst's point of view:

- Technologies vary from young adolescent to almost ageing, which makes current development activities rather different.
- Each modality has its own history resulting in independent design choices of programming languages, architectural concepts, tools etc.
- Architecture analysis issues include, among other things, migration monitoring, architecture-guided performance analysis, interface management, architecture verification, and cross-language reference analysis.

While differing on the one hand, modalities have common or at least related functionality on the other. So it is sensible to analyse and compare architectures of different modalities. To keep the analysis effort reasonable, a large set of

interoperating tools with limited tasks is necessary. These tools can be either self-made or provided by other organisations. A bottom-up approach to such an analysis framework seems most promising, i. e. learning from isolated problems and making their solutions part of a more generic tool-set.

12 An Examination of Evolution in Linux

Michael W. Godfrey, University of Waterloo, Canada

Recently, the reverse engineering research community has been investigating mechanisms for exchanging data and partial results between reverse engineering tools. Among the many subproblems inherent in implementing a standard exchange format is the design of schemas that represent views of source code at various levels of abstraction. In the first part of my talk, I presented some motivation for why the definition of an exchange format would be useful: I presented an analysis of the growth of the Linux kernel that shows that Linux has been growing at a geometric rate for several years. This is surprising, given previous formal studies of source code growth of large systems, and it seems to beg for a more detailed qualitative analysis of the evolution of Linux using a variety of anaylsis techniques and tools. In the remainder of my talk, I discussed some of the discussed some of the issues in defining, transforming, and exchanging "high-level" schemas, including previous work on the TAXFORM project, as well as recent discussions at the Workshop on Standard Exchange Format (WoSEF).

13 GraphXML – An XML-based graph description format

Ivan Herman, M. Scott Marshal, CWI Amsterdam, The Netherlands

GraphXML is a graph description language in XML that was developed as an interchange format for graph visualization. The generality and rich features of XML make it possible to define an interchange format that not only supports the

pure, mathematical description of a graph, but also the needs of information visualization applications that view graph-based data structures. A list of features supported by GraphXML includes: embedded and hyperlinked data references, hierarchical (nested) graphs, graph metadata such as graph theoretic properties, presentation styles such as color and geometry, and edit actions for graph transformations and history capture. This talk describes the collection of features and demonstrates them briefly using the open source graph visualization Java application "Royere". The focus of the talk is on the experience gained from the design that could be applicable to the GXL effort. The progress of the GraphML standardization effort, which was formed at Graph Drawing Symposium 2000 and includes a member of the GXL team, is also described. Royere is available at: `http://www.cwi.nl/InfoVisu/GVF/`

14 Reverse Engineering of C++ Programs

Rudolf Ferenc, Tibor Gyimóthy, University of Szeged, Hungary

One of the most critical issues in large-scale software development and maintenance is the rapidly growing size and complexity of the software systems. As a result of this rapid growth there is a need to understand the relationships between the different parts of a large system. We present a reverse engineering framework called Columbus that is able to analyze large C/C++ projects. Columbus supports project handling, data extraction, data representation and data storage. The source code extractor is capable for extracting the usual information such as the UML class model and the call graph; furthermore, its special feature is the handling of complex templates and their instantiation at source code level. Efficient filtering methods can be used to produce comprehensible diagrams from the extracted information, including the filtering on the visualized class models. The flexible architecture of the Columbus system (based on plug-ins) makes it a really versatile and an easily extendible tool for reverse engineering. A recently added feature which we are still working on is the capability to export the extracted data into our XML representation, which is called CPPML (C++ Markup Language).

15 Program Analysis Infrastructure

Jens Krinke, University of Passau, Germany

This talk argues that the purpose of interoperability of tools is not to reuse the data that tools produce but to reuse the infrastructure of that tools to produce that data. Instead of exchange data in a common format, the use of an API is sometimes possible and desirable: SNiFF+ is a source code engineering environment that allows accessing the internal repository via an API. The downside is that SNiFF+ doesn't store enough information to enable data flow analysis. Most reengineering tools don't use data flow information yet. However, in the future it will be necessary to use data flow analysis for specific reengineering tasks (like semantic preserving transformations of Java programs). Data flow analysis is expensive - in terms of time and memory consumption and in terms of building the infrastructure that does the analysis. At least the cost of building a new infrastructure can be avoided, as there are already a number of data flow analysis infrastructures and frameworks available. The talk presents the features and history of two successful infrastructures: Soot and SUIF.
There are some lessons learned from using data flow analysis for reengineering tasks:

- Data flow analysis based on ASTs is a bad idea (ASTs are not the right abstraction level and makes data flow analysis harder instead of easier).
- Infrastructure is expensive (even learning to use it).
- Choosing the right infrastructure is hard (evaluation means learning to use it).
- Compiler infrastructure tend to loose the connection to the source code, which is a problem for reengineering tasks.
- Academic infrastructure is fragile.
- Commercial interests might not allow using (the right) infrastructure.
- Standard infrastructure needs support and marketing.

With the view on infrastructure some problems of the XML approach are data explosion, performance and merging data - is the XML infrastructure good enough? The talk ends with the following suggestions:

- If you build new tools: will you probably need data flow?
- Don't build your own on top of ASTs, reuse infrastructure.

- Anticipate that your tool needs more data flow information than you think at the beginning.
- Define a schema and dump your fact storage in GXL, somebody might even find it useful!

16 Interoperability of Software Modification Tools

Chris Verhoef, Free University of Amsterdam, The Netherlands

In this talk I described that in the case of modifications tools, it is hard to rely on internal formats in the first place, let alone it would be possible to connect two heterogenous modification tools. Also there is no hope that one modification tool could ever be useful in another context.
Still the problem of parser sharing remains. Thats is: we should not re-implement parsers all the time. The solution for that is to share grammar specifications that are:

- void of idiosyncrasies (e.g. LR based rules)
- correct (i.e. tested on real code)
- complete (no missing language constraints)

Some the above requirements are feasible these days (see some other programs by me and others). This allows for a repository of grammar knowledge. Now this knowledge is the core of parser implementations/generations, which solves the bottleneck of the remaining parser problem.

17 Experiences in Tool Integration for Reengineering

Kenny Wong, University of Alberta, Canada

This talk summarizes the experiences and lessons learned from exploring and using a number of tool integration technologies to create more interoperable reverse engineering tools.

The Rigi graph visualization and editing tool uses RSF, a simple, lightweight, tuple-based graph file format for data integration. The advantages of RSF include ease of parsing, generation, editing, composition, and reading. Rigi uses Tcl/Tk for control and presentation integration. Users can write Tcl scripts to codify and automate reverse engineering activities.

In the RevEngE (Reverse Engineering Environment) project, the complementary capabilities of Rigi and the Software Refinery were integrated. The architectural analysis and visualization capabilities of Rigi were combined with the lower-level transformation and analysis capabilities of the Software Refinery over annotated abstract syntax trees. RevEngE used a message-based architecture with a repository based on the Telos conceptual modeling language and the ObjectStore object database. The repository used a global schema to integrate the tool data. Operational differences between the tools limited the usefulness of the environment.

In the Software Bookshelf project, an early prototype loosely integrated Rigi to a web-based user interface for software documentation. Selected program elements in the documentation have constructive views that can be executed by Rigi to provide dynamic content. In the underlying repository, references to data and tool capabilities were stored, not the actual data itself. Rigi ran as a client-side helper application in conjunction with a server-side CGI script.

Tool integration is one important strategy for designing and evolving more useful tools to users and to help address the tool adoption problem. These problems include aspects of relative advantage, complexity, trialability, and visibility. Compatibility is especially important for our tools with user comprehension strategies, preferred development tools, and forward engineering processes. Our approaches include: working on packaging issues, conducting user studies, identifying and streamlining common scenarios, focusing on lightweight technologies, and using benchmark applications for case studies. Also important is producing evidence of industrial success.

18 Interoperability of Re-Engineering Tools: A Requirements approach – what we have learnt so far

Kostas Kontogiannis, University of Waterloo, Canada

Tool interoperability for re-engnineering tools has emerged the last few years as a crucial area at research and practice due to the plethora of software analysis tools, CASE tools, and integrated developments environments (IDE's) that have appeared. It has become apparent that in order to facilitate collaboration between researchers and practitioners alike, we need to devise standards for different tools to exchange information pertaining to large information systems.
In this presentation, I discuss the problem form two angles. The first is the "de facto state of the art" and "state of practice" view drawn from past experiences and research efforts. This view is very helpful for assessing the lessons learnt, the benefits and the difficulties for having software re-engineering tools to inter-operate. The second view relates to where do we go from here. In this view, I will attempt to present tool interoperability as a layered reference model and as a list of functional and non-functional requirements that release to each layer in the reference model.

19 RDF and RDF Schema

Derek Rayside, University of Waterloo, Canada

A brief look at RDF and RDF Schema (both form W3C). Look at features for schema evolution and interoperability. Comparison with GXL, RSF and UML class diagrams.Problems of parametric types, co- and contra-variance. Inadequacy of graph theory to address typing problems. Do we want to model graphs? Or do we want a richer – e.g. Telos – modeling formalism?

20 Enabling Backtracking in Multi-Tool Reengineering Environments

Jens Jahnke, University of Victoria, Canada

Software Reverse Engineering (RE) and program understanding is a costly and human-intensive task. Many computer-supported tools have been developed in industry and academic to support humans in various RE activities. Today, several of these tools have reached a high level of maturity. It is often desirable to combine several tools into integrated reengineering environments. Unfortunately, currently existing tools use various incompatible information formats to store and exchange information. This impedes their integration in practice. Recently, researchers and practitioners have started a joint effort of defining a common exchange format based on XML technology. The naive approach of streaming GXL data from one tool to the next (pipeline architecture) bases severe problems in real-world scenarios. This is due to the fact that RE processes involve frequent backtracking to previous analysis steps. In this case the consistency with the abstraction derived before the backtracking step might be lost. Manually re-establishing this consistency is error-prone and requieres a lot of effort. An automatic consistency management based on the recorded derivation history can be used to automate the change propagation task. In my presentation, I have motivated and described a technique and implementation of such a consistency management component and service that can be reused for tool interoperability.

21 Integration Strategies for Building Software Exploration Tools

Margaret-Anne Storey, University of Victoria

In this talk I describe how data integration, control integration and presentation integration are used for visualizing software in the SHriMP system. Data integration is achieved by combining information extracted from public domain tools. Information presented includes architectural information, HTML'ized source code and documentation. Presentation and control integration are achieved by leverag-

ing the flexibility and extensibility of Java Beans. An architectural view provides placeholders for different views accessible at any level of granularity. Hypertext and zooming features support navigation between and across views. The talk concludes with a demonstration of using SHriMP for visualizing Java programs.

22 Format evolution

Ralf Laemmel, CWI and Free University of Amsterdam, The Netherlands

A systematic approach to the adaptation of XML documents and their DTDs is developed. The approach facilitates the evolution of XML-based formats. There are two essential ideas. Firstly, changes in the formats of documents can be often represented as stepwise transformations on the underlying DTDs. Secondly, the corresponding format migration of the original XML data is largely induced by the DTD transformations. We analyse the steps involved in such transformations and migration procedures.

23 From Research to Startup: Experiences in Interoperability

Arie van Deursen and Leon Moonen, CWI Amsterdam, The Netherlands

We present our experiences with tool interoperability by describing data exchange and coordination aspects of several projects performed at CWI over the last 10 years.
The first project is the development of the ASF+SDF MetaEnvironment, an environment for writing executable language specifications. This environment is developed as a set of cooperating heterogeneous tools that exchange data over a software bus called ToolBus. The coordination is defined using process algebra. Data is exchanged using a common annotated term format: ATerms. These ATerms employ maximal sharing of subterms to reduce the size of data to be

transmitted. This is especially important for the amounts of data involved in a typical legacy software renovation projects.
Other projects used this language technology as a basis for reverse and re-engineering research. We observed that standardizing grammars for well-known languages such as C or Cobol was hard to achieve, as different applications demand different ways of viewing such programs. Our best solution was to start a grammar base collecting all such grammars as opposed to attempts aiming at setting formal standards.
We have developed tools for language transformations and translation. Furthermore, we built a generic data flow framework that allows one to map languages to a common data flow language (Dhal). Data flow problems can be solved on Dhal and results are mapped back to the original language. This framework supports data flow analysis on systems written in multiple languages and allows for development of reusable libraries of DFA solutions. The main problem we have identified with these projects is divergence of language specifications because they are often refined when used to build different tools.
We have also connected several existing reverse engineering tools and components like databases and graph visualizers to parsers generated using the MetaEnvironment. Data exchange between these tools was done using an RSF like format. The coordination was done using scripting languages. Although this is a practical approach for building research prototypes, it is hard to create reusable industrial-strength tools this way.
To transfer our technology to industry, we have started the Software Improvement Group, a spinoff company that provides services in the areas of documentation generation, program transformation and strategic consultancy. The transformation tools are built using the MetaEnvironment and use the same methods of interoperability. The documentation generation tool DocGen is built around a repository. This repository is filled with artifacts that are extracted from legacy code using generated parsers. DocGen generates views on this repository in hypertext and graphs. Views are specified using a grammar from which we generate Java classes and visitors. These are used to query the database and traversed using the visitor to generate hypertext or graphs. Interoperability with DocGen is achieved by adhering to standards such as JDBC, http, and JavaDoc. Moreover, the data in the database or the views there on could be exported using a common exchange format such as GXL.

24 Data Interoperability for Software Reengineering Data

John Mylopoulos, University of Toronto, Canada

This workshop is looking at data interoperability solution that will make it possible to have reengineering tools exchange data. We note that there are different requirements for any proposed solution. If the solution is intended for research tools developed within the reengineering community the solution needs to be inexpensive and involve little overhead in adapting it. For industrial practice, on the other hand, the solution needs to scale up and allow for evolution.
The solution to the interoperability problem discussed in the workshop span a space between two extremes. At one extreme we have assembly time solutions where tools exchange data through customized interfaces. At the other extreme we have a repository based solution where all data are exchanged through a central repository. We note that the assembly line solution is best for situations which scalability and evolution are not requirements.
Finally, we note that database technolgy has advanced in the last 4 years with commercial repository products and XML servers which sit on top of a database engine. Moreover, much research is done on schema manipulation in support of data integration.

25 XML-Based Architecture-Exchange in Bauhaus

Holger Kienle, Rainer Koschke, and Erhard Plödereder University of Stuttgart, Germany

The talk, given by Holger Kienle, is in a nutshell an experience report with lots of 'X' acronyms–XML, XMI, GXL, and XSLT. The talk outlines how the Bauhaus reverse engineering system leverages XML-based technology for architecture-interchange and how it interfaces with UML tools. The talk discusses results of Gregor Schiele's Master Thesis "Beschreibung einer halb-automatisch abgeleiteten Architektur mit UML-Ausdrucksmitteln."
The talk is divided in three sections. The first section briefly introduces the Bauhaus reverse engineering tool and the concept of a resource graph. The next section discusses three methods to export the derived system-architecture from

Bauhaus. The most promising export to achieve interoperability with other tools is GXL, a standard exchange format for reengineering tools. The last section describes how we used the GXL output to realize a loose coupling between Bauhaus and a UML tool.

The Bauhaus tool facilitates program understanding of legacy code (currently C code only). The goal of the system is to assist the reengineer to derive the architecture of the system under examination. The code of the system is the only information that is employed for the reverse engineering process, since it is the only reliable source of information. In the reverse engineering process, the reengineer uses Bauhaus/Rigi, a graphical user interface that represents the facts of the source code in a graph structure. The nodes of the graph represent source-level entities (such as functions, types, and variables) as well as high-level architectural information (such as atomic components and subsystems). The edges of the graph describe relations between the nodes (such as calls between functions, and sets and uses of variables). The underlying data structure of the graph that is visualized is called the resource graph.

The reengineer can export the derived architecture (i.e., the resource graph, which conveys the architecture) in three different formats. The binary format (which is implemented by means of the Ada input/output library) is non-portable since it differs for different compilers and platforms, and can even differ for different versions of the same compiler. Hence, it is only useful (typically for a single user) to make the resource graph persistent. Because of this deficiency, Bauhaus offers a proprietary, ASCII-based exchange format. It is human-readable, avoids redundancy with string-sharing, and is line-oriented, which makes it is easy to parse and write. This format is primarily used for data exchange with the Fraunhofer IESE group. Lastly, we support the Graph Exchange Language (GXL), which is being developed as a cooperation of several universities. It is still under active development and discussion. Version 1.0 of GXL has been ratified by the Dagstuhl Seminar 01041. The format is based upon XML and proposed as a standard exchange format for reengineering tools.

One of the activities within the Bauhaus project is the UML integration of Bauhaus with a UML tool. Our motivation for providing this kind of interoperability is to provide a smooth introduction to Bauhaus for users that are already familiar with UML modeling elements and know how to operate a UML modeling tool. These users can work in their familiar environment and–in a perfect scenario–go back end forth between the two "worlds" of Bauhaus and UML. Bauhaus has features that a UML tool does not posses, such as semi-automatic analyses for recovering the system architecture, that makes it attractive for the user to employ Bauhaus for certain tasks in the architecture recovery process. Once the user becomes more familiar with Bauhaus, more and more tasks can be performed in Bauhaus.

We first considered a tight coupling between the tools. Tight coupling means that a change in one "world" is immediately reflected in the other one. For example,

if the user deletes an element in the UML tool, this change must be immediately reflected in Bauhaus's resource graph. It turned out that all the UML tools we looked at (Rational Rose, Together, and Innovator) were not suitable for this fine-grained coupling. For example, Together's API is unofficial and thus subject to changes and Innovator has only an API for its data repository. Thus, we considered loose coupling based on an exchange format. This approach has the distinct advantage, that potentially an arbitrary UML tool (e.g., the reengineers's favorite one) can be used. For the actual integration, a challenge that is not addressed in the talk is the finding of a good mapping of the system architecture between Bauhaus and UML. These two "worlds" are quite different, Bauhaus using a resource graph (which is based on C) and UML using a graphical notation rooted in object-oriented design. The rest of the talk addresses the other challenge, namely to realize the actual data exchange.
Stream-based exchange of models between UML tools can be achieved with XMI, which is tool independent and based upon XML. Currently UML tools do not fully support XMI, suffer from bugs in the implementation, and generate rather huge output files. On the Bauhaus side, the GXL export realizes the loose coupling. The actual data exchange works as follows: The Bauhaus resource graph is exported with GXL. A translator maps the GXL representation into XMI, which in turn can be read in by a UML tool. The translation process cannot achieve a one-to-one mapping. Hence, information will be lost. The reverse data exchange from UML to Bauhaus works analogously. The reverse mapping loses information as well. For the actual translation, two alternatives are considered. Since both GXL and XMI are XML documents, XSLT can be employed for the transformation. We started to implement prototypical XSLT-scripts for the translation process, but the translation had a disastrous performance. The other alternative employs the Xerces XML-library for Java, which provides APIs for parsing XML. The actual translation is now coded in Java. GXL is parsed with the Xerces library and an in-memory representation of the resource graph in Java is built. This representation is translated in-memory to the corresponding UML model and finally written out as XMI file. The performance turned out to be acceptable, however, we had to use SAX (the low-level API) instead of DOM (the high-level API). DOM constructs the whole parse tree of the XML document in memory, which turned out to be too huge for our input files.
We hope to employ GXL to exchange architecture data with other reengineering tools in the future. A first step to simplify this process is the definition of standard schemas for GXL. An effort to define low-level, mid-level, and high-level schemas is currently underway.
Two main conclusions can be drawn from our experiences. Firstly, GXL proofed its usefulness for tool integration. Secondly, XMI-based data-exchange is feasible even though the current UML tools are not yet mature.

26 Low Level Schema breakout session

Timothy Lethbridge, Rudolf Ferenc, et al.

This breakout session tried to agree upon a working schedule to achieve a common C++ schema at AST (abstract syntax tree) level. A brief overview over the results was given at the end of the seminar.

27 Ratification of GXL 1.0

Susan Elliot Sim et al.

As one big and important result of this Dagstuhl Seminar on Interoperability of Reverse Engineering Tools, the reverse engineering community agreed upon the GXL format (Graph eXchange Language) as standard interchange format. Version 1.0 of GXL was ratified by the participants. For more information, please refer to the GXL web site at `http://www.gupro.de/GXL`