Towards Specifying Reengineering Services Using Graph-based Schemas

Jürgen Ebert, Andreas Winter
University of Koblenz, Germany
{ebert|winter}@uni-koblenz.de

August 2003

1 *GUPRO*

GUPRO (Generic Understanding of Programs) is a generic tool for supporting querying and browsing of software source code. An overview of *GUPRO*, as well as hints for further reading, can be found in [EKRW02].

GUPRO is generic in the sense, that heterogenous software written in different languages can be treated on arbitrary levels of abstraction. The key concept to support this genericity is the use of conceptual schemas to define the appropriate conceptual view on sofware.

To use *GUPRO*, appropriate extraction tools (e.g. light-weight parsers) must be supplied, which extract the facts from a given software into the graph-based *GUPRO* repository according to the respective schema. These facts are analyzed by graph algorithms and GReQL (Graph Repository Query Language) graph queries.

Using appropriate extractors, the repository content can also be viewed by the generic *GUPRO* browser, which allows multidimensional browsing through the sources. Browsing is also guided by the given schema. Furthermore, to handle C or C++ source code, preprocessor facilities are included into the browser. This facility allows browsing on any level of detail arbitrarily chosen between perprocessor input and preprocessor output.

The generic browser and the generic query component are fully integrated with each other. Thus, a selection of code in the browser can be used as a starting point for query. The output of a query can be visualized as a table containing hyperlinks or as a selection in the browser window.

GUPRO heavily relies on querying the graph-based repository as its enabling technology. Also, a script language *clg* (Command Line GReQL) is supplied which allows to execute stored query suits for standard applications.

2 Foundations

In *GUPRO* all conceptual views as well as the implementation are based on graph technology, e.g. the source code is represented as a TGraph in the repository. TGraphs are typed, attributed, and ordered directed graphs. Since vertex and edge types allow multiple inheritance and since the attribute types include all relevant basic types and type constructors, TGraphs are a quite generally usable multipurpose graph class.

TGraphs as a repository structure are supported by the GraLab class libraries which are supplied for C++ and for Java. GraLab also includes GReQL query facilities and provides integration of graph algorithms and graph querying. Thus, all programming in the *GUPRO* context can be done directly in a seamless manner on the conceptual graphs used.

Classes of TGraphs are specified using extended entity relationship schemas (EER) which are nowadays visualized using the UML-notation. Additional integrity constraints are expressed using the Z-like constraint specification languages GRAL (GRAph specification Language). More information on the EER/GRAL approach on graph-based modeling is given in [EWD$^+$96].

TGraphs cover the graph part of the more general hierarchical hypergraph concept of GXL (Graph Exchange Language). Thus, TGraphs form an ideal basis for exchanging reengineering information between tools. All TGraphs can easily be exchanged using GXL.

Based on GXL several interoperations between *GUPRO* and other tools have been instantiated meanwhile. *GUPRO* has been plugged together with extractor frontends like Gyimothi's Columbus or Sneed's analysis tools, for instance. In cooperative analysis tasks *GUPRO* was combined with Holt's grok/TA graphs, and Koschke's Bauhaus. daVinci and GraphViz have been combined with *GUPRO* for visualization purposes.

While source languages to be handled by the *GUPRO* environment must be translated into TGraphs by some graph extraction tool, the kind of extraction depends on the graph schema. In *GUPRO* several graph schemas for different languages have been collected. These schemas introduce a further level of coupling between tools when they are used as a means of communication. Since GXL also supports the exchange of schemas and since *GUPRO* schemas are a subclass of GXL schemas, interoperability is also supported on the schema level.

3 Lessons learned and further questions

Developing appropriate conceptual schemas for different languages and for different levels of abstraction is an exhaustive and costly task. It affords a lot of conceptual knowledge about languages, much communication between researchers, and consideration of the reengineering tasks to be addressed. On the other hand, we learned that interoperability on schema level is manageable and useful, though also here adaptations are to be done.

If interoperability of tools is going to be supported in more depth the derivation of reference schemas for different languages and on different levels of abstraction is inevitable. The work started by the Dagstuhl seminar 461 in January 2001 will have to be finished by a *collection of reference schemas* at least on the syntactic level, the middle level, and the architecture level. Furthermore the adaption of these reference schemas to different languages has to be given, too.

Since the formal adaptation of (reference) schemas to concrete applications is an important vehicle for writing generic tools and for supplying the interoperability context for different tools, an easy-to-use and precisely-defined *mechanism for schema integration and transformation* has to be developed. This task might be tackled by a schema algebra approach which contains all important operations like e.g. adding, deleting, generalizing, specializing, lifting, and lowering of graph elements, as well as appropriate abstraction concepts.

Furthermore, conceptual schemas and elementary *reengineering services* have to be combined more closely. They must no longer be seen as different concerns. In the same way as source language descriptions are being developed and combined using small or medium size schema fragments, also the functionality of reengineering tools has to be broken down into small or medium size *service fragments*.

The approach of modeling reengineering data by conceptual schemas and representing the data by TGraphs can also be applied to *dynamic data*. Here, further tasks can be identified for schema definition, adaptation and transformation as well as for service definition.

The same applies also if higher level description documents in *visual languages* are to be included into a round-trip reengineering approach or if aspect recognition and extraction are to be tackled.

The specification of service fragments for static and dynamic reengineering and their embedding into software development processes can be done to a large extent using the schema algebra approach from above.

4 Conclusion

Schemas define data structures for reengineering techniques and provide a proper means for specifying the interoperability context to enable seamless tool interoperability.

Reengineering services are built upon these schemas and use different tools and approaches. The development of services and their appropriate schemas has to be based on a precisely defined mechanism for schema integration and transformation. Creation of reengineering services is enabled by adapting and plugging together reference services to form a comprehensive set of reengineering services.

References

[EKRW02] Jürgen Ebert, Bernt Kullbach, Volker Riediger, and Andreas Winter. GUPRO - Generic Understanding of Programs - An Overview. *Electronic Notes in Theoretical Computer Science,* 72(2), 2002.

[EWD+96] Jürgen Ebert, Andreas Winter, Peter Dahm, Angelika Franzke, and Roger Süttenbach. Graph Based Modeling and Implementation with EER/GRAL. In Bernhard Thalheim, editor, *Conceptual Modeling - ER'96*, pages 163–178. Springer, 1996. LNCS 1157.