A Summary of: Rule Execution in Graph-Based Incremental Interactive Integration Tools

Simon M. Becker and Sebastian Lohmann and Bernhard Westfechtel
Department of Computer Science III, RWTH Aachen University
Ahornstraße 55, D-52074 Aachen, Germany

Abstract. *Development processes in engineering disciplines are inherently complex. Throughout the development process, different kinds of inter-dependent design documents are created which have to be kept consistent with each other. Graph transformations are well suited for modeling the operations provided for maintaining inter-document consistency. In this summary, we describe a novel approach to rule execution for graph-based integration tools operating incrementally and supporting conflict detection and user interaction. A full version of the paper is available as [BLW04].*

Keywords: incremental integration, graph transformation

1 Introduction

In *development processes* in engineering disciplines, different kinds of *documents* are created. These documents are inter-dependent regarding their contents and as a consequence, have to be kept consistent with each other. For example, in software engineering the source code of a software system must match its high-level description in the software architecture.

Current support for ensuring documents' consistency is mainly transformation oriented, allowing the automatic generation of a target document from a source document. But in real-life development processes, transformations are ambiguous or incomplete, requiring resolution by user interactions. If approaches like concurrent and simultaneous engineering are applied, dependent documents are often modified simultaneously. Therefore, incremental change propagation is needed to restore consistency without loosing modifications made to one of the documents. Nevertheless, because developers have to be able to work independently from each other, change propagation has to be decoupled from modifications to some extent. Changes cannot be propagated at once to all dependent documents. Instead, there are discrete points in time where developers decide to perform a synchronization of their documents.

In such a setting, there is a need for incremental and interactive *integration tools* for supporting inter-document consistency maintenance. An integration tool has to manage *links* between parts of inter-dependent documents. These parts are called *increments* in the sequel. The tool assists the user in *browsing* (traversing the links in order to navigate between related increments in different documents), *consistency analysis* (concerning the relationships between the documents' contents), and *transformations* (of the increments contained in one document into corresponding increments of the related document).

Graphs and graph transformations have been used successfully for the specification and realization of integration tools [dLV02, BMP02]. However, in the case of incremental and interactive integration tools specific requirements have to be met concerning the execution of integration rules. In this paper, we describe a novel approach to rule execution for graph-based integration tools operating incrementally and interactively which is based on triple graph grammars [Sch95]. Rather than executing a rule in atomic way, rule execution is broken up into multiple phases. In this way, the user of an integration tool may be informed about all potential rule applications and their mutual conflicts so that (s)he may take a judicious decision how to proceed.

2 Graph-Based Specification of Integration Tools

In complex scenarios as described in the previous section, an integration tool needs to maintain a data structure storing links between inter-dependent documents. This data structure is called integration document. Altogether, there are three documents involved: the *source document*, the *target document*, and the *integration document*. Please note that the terms "source" and "target" denote distinct ends of the integration relationship between the documents, but this does not necessarily imply a unique direction of transformation.

All involved documents may be modeled as graphs, which are called *source graph*, *target graph*, and *correspondence graph*, respectively. Moreover, the operations performed by the respective tools may be modeled by graph transformations. *Triple graph grammars* [Sch95] were developed for the high-level specification of graph-based integration tools. The core idea behind triple graph grammars is to specify the relationships between source, target, and correspondence graphs by *triple rules*. A triple rule defines a coupling of three rules operating on source, target, and correspondence graph, respectively. By applying triple rules, we may modify coupled graphs synchronously, taking their mutual relationships into account.

As already explained earlier, we cannot assume in general that all participating documents may be modified synchronously. In case of asynchronous modifications, a triple rule is not ready for use. However, we may derive *asynchronous rules* from the synchronous rule in the following ways: A *forward rule* assumes that the source graph has been extended, and extends the correspondence graph and the target graph accordingly. Analogously, a *backward rule* is used to describe a transformation in the reverse direction. Finally, a *consistency analysis* rule is used when both documents have been modified in parallel. In this case, only the correspondence graph has to be updated.

Unfortunately, even these rules are not ready for use in an integration tool as described in the previous section. In the case of non-deterministic transformations between inter-dependent documents, it is crucial that the user is made aware of conflicts between applicable rules. Thus, we have to consider all applicable rules and their mutual conflicts before selecting a rule for execution. To achieve this, we have to give up *atomic rule execution*, i.e., we have to decouple pattern matching from graph transformation.

3 Rule Execution

Each integration rule is automatically translated into a set of graph transformations. These rule specific transformations are executed together with some generic ones following an *integration algorithm*. Here, we present a short overview of the algorithm only. A detailed description can be found in [BLW04].

The increments contained in integration rules may have different roles affecting how they are treated by the algorithm: increments can be dominant, normal, or context increments. Each increment in source or target graph may be referenced by at most one link as dominant or normal increment created by exactly one rule, whereas each increment may be a context increment for an arbitrary number of links. To facilitate pattern matching, dominant increments serve as starting point. Context increments are used to embed edges during transformations.

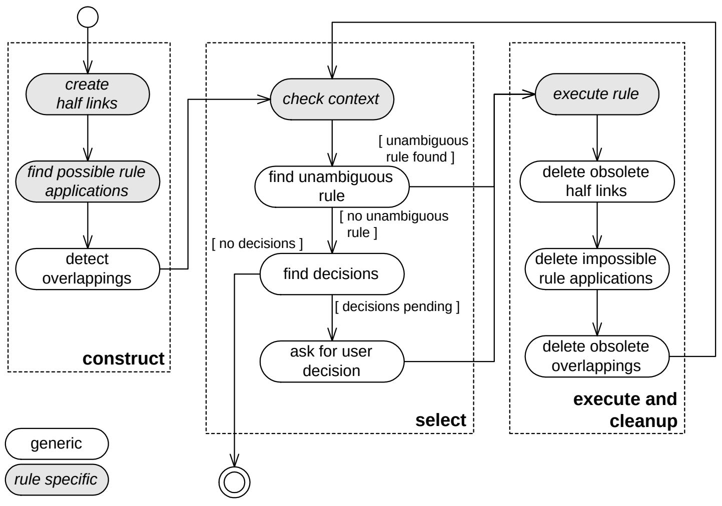

Figure 1: Integration algorithm

Figure 1 shows a UML activity diagram depicting the integration algorithm. To perform each activity, one ore more graph transformations are executed. Activities that require the execution of rule specific transformations are marked grey and italic. The overall algorithm is divided into three phases.

During the first phase (construct), all possible rule applications and conflicts between them are determined and stored in the graph. First, for each increment in the source document that has a type compatible to the dominant increment's type of any rule, a half link is created that references this increment.

Then, for each half link the possible rule applications are determined by matching the increments contained on the left-hand side of the corresponding forward, backward, or correspondency analysis rule, that are non-context increments. The result of the pattern matching is explicitly stored in the correspondence graph. This is necessary to decouple the matching and execution of rules. The context increments are matched in the subsequent phase of the algorithm. The last step of this phase is a generic transformation detecting and marking overlappings between possible rule applications.

While the construct phase is executed only once, the following two phases are executed in a loop until the integration is complete. In the select phase, for all rule applications the context is checked by matching all context increments of the rules. Again, the matching is stored in the correspondence graph. If one rule application, whose context is present, is unambiguous, it is automatically selected for execution. Otherwise, the user is asked to select one rule among the rules with existing context. If there are no executable rules, the algorithm ends.

In the last phase (execute and cleanup), the selected rule is executed using the increments previously matched. For forward (backward) rules, the non-context increments in the target (source) document are created and a new link is established that references source and target increments. When executing a consistency analysis rule, the link is created between existing increments. After that, some operations are performed to adapt the information that was collected in the construct phase to the new situation. For example, possible rule applications are deleted, that have become impossible after the execution of a conflicting rule.

The algorithm continues by going back to the select phase. The context has to be checked again because previously missing context increments may have been created by the preceding rule execution. The loop continues until the integration is finished.

4 Conclusion

We have presented a novel approach to the execution of integration tools in incremental and interactive integration tools using graph transformations. We have realized this approach, in a research prototype called *IREEN*, an *I*ntegration *R*ule *E*valuation *EN*vironment. In an industrial cooperation with the German software company Innotec the approach was evaluated with a simplified prototype for the integration of flow sheets and simulation models implemented in C++. Experiments with the prototype showed that our approach considerably leverages the task of keeping dependent documents consistent to each other.

References

[BLW04] Simon M. Becker, Sebastian Lohmann, and Bernhard Westfechtel. Rule execution in graph-based incremental interactive integration tools. In *Proc. of the 2nd International Conference on Graph Transformations (ICGT 2004)*, LNCS 3256, pages 22–38. Springer, 2004.

[BMP02] L. Baresi, M. Mauri, and M. Pezzè. PLCTools: Graph transformation meets PLC design. *Electronic Notes in Theoretical Computer Science*, 72(2), 2002.

[dLV02] J. de Lara and H. Vangheluwe. Computer aided multi-paradigm modelling to process petri-nets and statecharts. In *Proc. of 1st Int. Conf. on Graph Transformations (ICGT 2002)*, LNCS 2505, pages 239–253. Springer, 2002.

[Sch95] Andy Schürr. Specification of graph translators with triple graph grammars. In *Proc. of the 20th Intl. Workshop on Graph-Theoretic Concepts in Computer Science (WG 1994)*, LNCS 903, pages 151–163, Herrsching, Germany, 1995. Springer.

The “Software Analysis and Verification” Column

Jens Knoop *
*Institute of Computer Languages, Vienna University of Technology

1 Software Analysis and Verification: A New Column of the EASST Newsletter

Software analysis and verification, and *software transformation* are as closely related as Siamese twins. Indeed, before undergoing a (non-trivial) transformation virtually every piece of software will beforehand be subject to some analysis or verification in order to ensure the applicability of the transformation, to ensure at a minimum that the application of the transformation will not cause any harm. More generally than suggested by the title of this column, the software analysis and verification column seeks contributions on the theory and practice of methods for the analysis, verification, and transformation of software. Contributions, which are related to at least one of these topics are welcome. Especially welcome are contributions bridging two of these topics such as analysis and transformation. Additionally, and without being limited to, also position papers, book reviews, announcements of forthcoming events such as conferences and workshops, or reports on past conferences are welcome, too.

Contributions to this column should directly be sent to the column editor, preferably by email to `knoop@complang.tuwien.ac.at`, or by letter post to:

Univ.-Prof. Dr. Jens Knoop
Institute of Computer Languages
Vienna University of Technology
Argentinierstr. 8 / E1851
1040 Vienna, Austria
`knoop@complang.tuwien.ac.at`

In this first issue of this new column we report on three workshops and symposiums within the scope of the software and analysis column: ISoLA 2004, SYNASC 2004, and ASM 2004. It should be noted that the selection of these three meetings for presentation here is biased by recent involvements of the column editor as a member of the Programme Committee (ISoLA 2004, SYNASC 2004) and invited speaker (SYNASC 2004, ASM 2004). . . . .

1.1 Report on ISoLA 2004

ISoLA 2004 was the inaugural instance of the *International Symposium on Leveraging Applications of Formal Methods* (*ISoLA 2004*). It took place from October 30th to November 2nd in Paphos, Cyprus. Initiated and organized by the current EASST President Tiziana Margaria (General Chair), Bernhard Steffen (Programme Chair) and Anna Philippou (Local Chair), ISoLA 2004 attracted some 70 scientists and researchers from around the world from both academia and industry.

This is in line with the intent of this new symposium series, which, as expressed in the call for papers and the preface of the symposium proceedings, is "*to bridge the gap between designers and developers of rigorous tools, and users in engineering and in other disciplines, and to foster and exploit synergetic relationships among scientists, engineers, software developers, decision makers, and other critical thinkers.*"

This first instance of ISoLA featured 5 sessions with 15 presentations of contributed research papers, a poster session, a panel on "Formal Approaches to Complex Software Systems" moderated by Jose-Luis Fernandez-Villacanas-Martin (Commission of the European Union), and 13 Thematic Sessions on specific hot topics of special relevance and interest to the formal methods developers' and users' community. These sessions were composed of 33 presentations, which were solicited by the respective thematic session chairs, who also acted as a shepherd of these contributions. A further highlight of the symposium was the invited lecture delivered by David Harel (Weizman Institute, Israel), which was entitled "Towards an Odor Communication and Synthesis System."

The topics of the regular and thematic sessions covered a spectrum, which ranged from model checking and validation over scheduling and performance issues of real-time embedded systems to component-based and networked applications and the usage of formal methods in industry.

Of a special interest with respect to the scope of this column was the Thematic Session on "Program Analysis and Transformation." This session was devoted to discussing the state-of-the-art and to further identifying the most urgent challenges related to beneficially using formal methods in program analysis and transformation, e.g. in verifying optimizing compilation.

The thematic session on program analysis and transformation featured three presentations, which approached the session's topic from quite diverse perspectives, and served as the starting point for further in-depth discussions at the symposium. They were delivered by Dan Quinlan (Lawrence Livermore National Laboratory, CA) on the "Classification and Utilization of Abstractions for Optimization," Wolf Zimmermann (University of Halle-Wittenberg, Germany) on the "Correctness of Transformations in Compiler Back-Ends," and Byron Cook (Microsoft Research, UK) on "Finding API Usage Rule Violations in Windows Device Drivers Using Static Driver Verifier."

The preliminary proceedings of this symposium are available as a technical report of the Department of Computer Science of the University of Cyprus [MSPR04]. It is planned to publish selected resubmitted papers in post conference proceedings and in special issues of distinguished journals.

In 2006 it is planned to return to Paphos, Cyprus, for the second large event of ISoLA. In 2005, there will be a thematically focused event, which will be organized in North-America, most likely in the Washington D.C. area.

Further information on the meeting in Paphos can be found on the Web page of the symposium at `http://sttt.cs.uni-dortmund.de/isola2004/dflt␣ns/index.html` . . . .

1.2 Report on SYNASC 2004

From September 26th to September 30th, the *6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing* (*SYNASC 2004*) was held in Timişoara, Romania. The symposium was hosted by the University of the West Timişoara, and it was jointly organized by its Department of Computer Science, the Research Institute for Symbolic Computation of the Johannes Kepler University of Linz, Austria, and the Institute e-Austria Timişoara. Founded in October 2002, the Institute e-Austria Timişoara is jointly funded by the governments of Romania and Austria to further strengthen the ties and research collaborations between Romania and Austria, both scientifically and economically, and to create a gate for Austrian IT companies to Romania and further to other countries in Eastern Europe, certainly something of interest in its own for an association carrying the acronym EASST: more information on the Institute e-Austria Timişoara and its mission can be found at `http://www.ieat.ro/`.

The Program Co-Chairs of SYNASC 2004 were Viorel Negru (West University, Romania) and Tudor Jebelean (University of Linz, Austria), the General Co-Chairs Ştefan Măruşter (West University, Romania), and Bruno Buchberger (University of Linz, Austria), and the Local Co-Chairs were Daniela Zaharie (West University, Romania) and Dana Petcu (Institute e-Austria, Romania).

This year, SYNASC featured 29 presentations of contributed research papers and 4 invited lectures, several of which were of special interest with respect to the scope of this column. Among these, also three of the invited presentations by Gabriel Ciobanu (Romanian Academy, Iaşi) on “Simulation and Verification of the Biomolecular Systems”, by Tetsuo Ida (University of Tsukuba, Japan) on “Layers of Abstraction in Symbolic Computation,” and by Stephen M. Watt (University of Western Ontario London, Canada) on “Optimizing Compilation for Symbolic-Numeric Computation.”

Co-located with SYNASC 2004, there were four workshops on *Agents for Complex Systems* (*ACSYS 2004*), *Computer Aided Verification of Information Systems* (*CAVIS 2004*), on *Symbolic Grid Computing* (*SGC 2004*), and the *Natural Computing Workshop* (*NCW 2004*). The joint proceedings of SYNASC 2004 and the affiliated workshops are available through your local book seller [PNZJ04]. Selected resubmitted papers will later be published in a special issue of the journal “Annals of the University of Timisoara,” ISSN 1224-970X.

Further information on SYNASC 2004 and on the SYNASC Symposium series can be found on the SYNASC 2004 web page at `http://synasc04.info.uvt.ro`. ....

1.3 Report on ASM 2004

Wittenberg, Germany, home to Martin Luther, and thus also known as Lutherstadt Wittenberg, was the venue of the *11th International Workshop on Abstract State Machines* (*ASM 2004*). The workshop took place from May 24th to May 28th. It was organized by Wolf Zimmermann (University of Halle-Wittenberg, Germany), who together with Bernhard Thalheim (University of Kiel, Germany) also served as a Program Co-Chair.

The ASM workshop series is devoted to the dissemination and promotion of advances in theory, practice, and applications of ASMs and ASM methods. Typical topics of interest include the high-level design, analysis, validation, and verification of computing systems. This year, the program was composed of 11 contributed research papers and 6 invited presentations, which were delivered by Yuri Gurevich

(Microsoft Research, USA), Hans-Michael Hanisch (Universität Halle-Wittenberg, Germany), Jan Van den Bussche (Limburgs Universitair Centrum, Belgium), Jens Knoop (TU Wien, Austria), Egon Börger (Universitá di Pisa, Italy), and Hans Langmaack (Universität Kiel, Germany). The spectrum of topics covered by the presentations at the workshop ranged from temporal verification of monodic ASMs over ASM semantics for SSA intermediate program representations to ASM specifications of C# threads.

The proceedings of the workshop have been published in the LNCS series of Springer Verlag, volume number 3052 [ZT04]. A companion technical report published by the University of Halle-Wittenberg contains the abstracts of industrial experience reports, tool demonstrations, and work in progress papers, which have also been presented at the workshop.

Further information on ASM 2004 and the ASM Workshop series in general can be found on the ASM 2004 web page at `http://swt.informatik.uni-halle.de/ASM2004/`.

ASM 2005 is going to take place from March 8th to 11th, 2005, in Paris, France. The submission deadline for papers is on January 10th, 2005. The full call for papers can be found at *http://www.univ-paris12.fr/lacl/Asm05/*. ....

References

[MSPR04] Tiziana Margaria, Bernhard Steffen, Anna Philippou, and Manfred Reitenspiess, editors. *International Symposium on Leveraging Applications of Formal Methods* (*ISoLA 2004*), TR-2004-6, 2004. Preliminary Proceedings.

[PNZJ04] Daniela Petcu, Viorel Negru, Daniela Zaharie, and Tudor Jebelean, editors. *6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing* (*SYNASC 2004*), 2004. ISBN 973-661-441-7.

[ZT04] Wolf Zimmermann and Bernd Thalheim, editors. *11th International Workshop on Abstract State Machines* (*ASM 2004*), volume 3052 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.