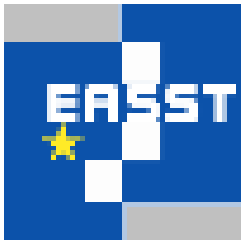
THE EASST NEWSLETTER

Volume 9

December 2004



European Association of Software Science and Technology

Editor:
Dr. Julia Padberg
Technische Universität Berlin
FB 13, Sekr. FR 6-1, Franklinstr. 28/29, D-10587 Berlin

E-mail : padberg@cs.tu-berlin.de
URL : http://tfs.cs.tu-berlin.de/~padberg/
Tel : +49-30/314-24165
FAX : +49-30/314-23516

Contents:

Welcome!

Dear EASST Members,

again I hope to present an interesting and worthwhile Newsletter.

First I want to congratulate the winners of the EASST-awards. At the ICGT04 Simon M. Becker, Sebastian Lohmann, and Bernhard Westfechtel received an EASST award for their paper *Rule Execution in Graph-Based Incremental Interactive Integration Tools*. In this volume we present a summary of the award-winning paper. At FMICS the EASST award has been given to Martin Fränzle and Christian Herde for their paper on HySat, a new bounded model checker for linear hybrid systems. The title is *Efficient Proof Engines for Bounded Model Checking of Hybrid Systems* and the paper can be found at http://www.imm.dtu.dk/%7Emf/FMICS04_Fraenzle_Herde.pdf.
I am glad that the scope of relevant topics in the EASST newsletter has been further broadened: We have a new column on Software Analysis and Verification that is edited by Jens Knoop. He introduces THE SOFTWARE ANALYSIS AND VERIFICATION COLUMN and reports on three related events. Additionally, we have a lot of interesting contributions in the other columns.

Moreover, we have the reports from FMICS as well as from ICGT 2004, so those who could not attend know what they have missed.

Our president Prof. Tiziana Margaria has taken up her new position at the University of Göttingen and now has the chair "Service Engineering for Distributed Systems". I hope she has a good start, a lot of success, and still time for EASST.
And I am happy and proud to announce the birth of my second son Kilian Padberg on June 3, 2004. So again I am in maternity leave....but the EASST newsletter will continue to come to you.

Yours sincerely,

Julia Padberg
(EASST-Secretary)

PS: Again, if you have any contribution you would like to have in the forthcoming newsletter, please send it to me.

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.

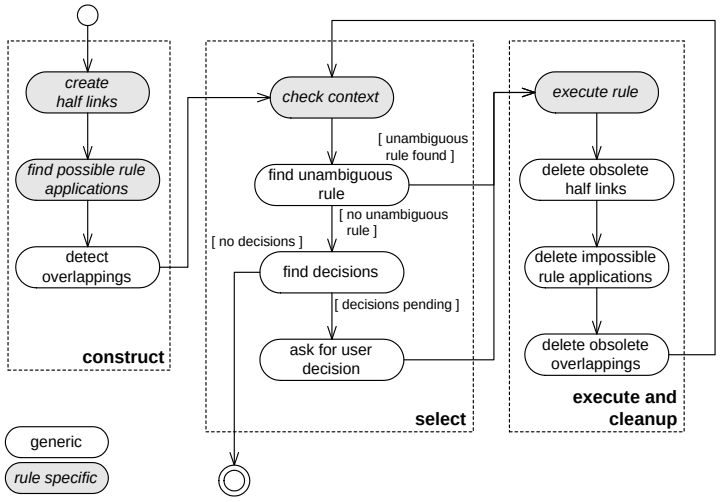


Figure 1: Integration algorithm

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.

The Column on Visual Modelling Techniques

News on the SegraVis Research Training Network

Edited by Reiko Heckel *
*University of Leicester, UK

Abstract. *In this second Column on Visual Modeling Techniques we continue our presentation of the partners of the Research Training Network SegraVis on* Syntactic and Semantic Integration of Visual Modeling Techniques *and report on some recent events related to the network.*

Keywords: visual modeling techniques, SegraVis sites and events

1 Halftime

End of September the network completed its second out of four years. To that date, its 12 nodes hosted 20 young researchers for a total of about 100 months. Network members (co-)organized more than 30 scientific events, like conferences, workshops and seminars, and about 20 training events including tutorials, introductory and a school. The midterm report, which is online at `www.segravis.org` mentions about 60 joint publications of authors from different sites of the network.

The occasion was celebrated at the midterm review meeting in Rome on 2nd October 2004, along with the International Conference on Graph Transformation, itself one of the major network events locally organized by Paolo Bottoni. In a pleasant place in the center of the Eternal City, and supported by perfect food and great weather, the meeting contained presentations of the network coordinator, the scientific officer, the 12 site coordinator, and 16 SegraVis grant holders.

SegraVis is a European Research Training Network under the Fifth Framework, running from October 2002 until September 2006. Besides a collaborative research project, the network offers grants to young researchers.

Enquiries for open positions, ***especially at the post-doc level****, can be sent to the local coordinators or the network manager* ***at all times****, see* `www.segravis.org` *for details.*

In this newsletter we will continue our introduction of network members and report on the co-located conferences VL/HCC and ICGT as well as some of their satellite events.

2 Members

Let us introduce in more detail three more of the 12 sites of the network with their key personnel and research interests.

Berlin

The SegraVis team at the Technical University of Berlin consists of two research groups: the group on Theoretical Computer Science/ Formal Specifications headed by Hartmut Ehrig and Gabriele Taentzer, and the group on Computer-Supported Information Systems led by Herbert Weber and Ralf Kutsche. Both groups have considerable experience in foundations and practice of object-oriented software development, especially in UML-based modelling and meta-modelling.

The group of Hartmut Ehrig is well-known for its work on graph transformation, including its theory as well as the implementation of concepts and results in AGG, a development environment for graph transformation systems. The group considers the precise definition of visual languages as main application area for graph transformation and develops tools for defining and generating environments for visual languages. This development is based on AGG and uses Eclipse and several Eclipse plug-ins. The focus is on problems like diagram parsing, model transformation, and integrated management of syntactic and semantic aspects.

As primary example for visual languages, the UML is considered. The experience with formal specification languages is used to analyze the UML semantically in order to improve and extend the formal foundation of the UML, making it a more precise software engineering tool. Together with a group of mechanical engineers from the University of Stuttgart, this work is performed in the context of a Priority Program of the German Research Council (DFG) on the Integration of Software Specification Techniques, headed by Hartmut Ehrig. This line of research focusses on the integration of object oriented software specification techniques with applications in engineering, in particular in production automation. Furthermore, a visualization for the textual language OCL (being part of the UML) has been developed, as a new visual language.

The group of Herbert Weber has practical experience in modelling complex software systems with object-oriented languages and methodologies (like OMT and UML) and in model-based approaches to software engineering, following the paradigm of Continuous Software Engineering, which is related to the OMG's MDA activities. The methodologies and techniques have been developed in the context of several application projects in environmental information systems, health care and air traffic control — and continuously improved in cooperation with our associated research institute Fraunhofer ISST, with industrial partners, and with research groups all over the world. Currently they are applied in a large project, the European Migration Network, an information system integrating heterogeneous data sources on migration and asylum all over Europe.

Canterbury

The SegraVis team at the University of Kent is coordinated by Dr. Peter Rodgers and consists of members of the Software Engineering Research Group and the Applied and Interdisciplinary Research Group.

It encompasses interests in automatic layout of software engineering models, model driven engineering, model transformation systems and visual alternatives to textual modelling notations.

Within the topic of automatic layout of software engineering models, the team is developing general graph drawing tools and systems with models derived from reverse engineering and user developed UML models as target application areas. The model driven engineering work includes implementation of the KMF (Kent Modelling Framework) by Octavian Patrascoiu and David Akehurst, a test bed for further research in meta modelling.

Integrated into the KMF is YATL (Yet Another Transformation Language), designed to facilitate the mapping between models and between models and code. The team is also heavily involved in developing Constraint Diagrams, a visual alternative for OCL (Object Constraint Language). This work is important because OCL is currently the only remaining textual language in the UML.

Leiden

The SegraVis team in Leiden is affiliated with the Leiden Institute of Advanced Computer Science (LIACS), the computer science research institute of the Faculty of Mathematics and Natural Sciences of Leiden University. The work is carried out in the Algorithms and Foundations of Software Technology groups, prominent members of which are the professors Grzegorz Rozenberg, Joost Kok, Thomas Baeck, Luuk Groenewegen and Farhad Arbab.

Our main areas of research are:

- *Molecular computing* in particular, the suitability of biomolecules for computing as well as theoretical models for various aspects of molecular computing.
- *The theory of formal languages, graphs and graph transformations, and concurrent processes*, in particular such fields as: grammars, automata and transducers, combinatorial structure of formal languages; models for natural computation, node-rewriting graph grammars, hypergraph grammars, Petri nets, traces, synthesis of concurrent systems from the specification of their behaviour.
- *Coordination*, studying how complex systems can be constructed from components distinguishing individual entities and their coordinated interaction, as well as their integration into environments which can range from technical (other software or hardware) over organisational (people, teams, departments) to natural (environmental, biological or biochemical) ones.
- *Optimization*, focussing on algorithms for Simulated Evolution, in particular Evolutionary Computation.
- *Data Mining*, concerned with mining (semi-)structured data, in particular graph mining, and with data mining problems from the area of Natural Computation.

3 Events

VL/HCC 2004

Visual Language and Human Centric Computing was the 20th edition in the Visual Languages Conference Series of the IEEE, held in Rome between the 27th and 29th September 2004. It was preceded by two tutorials: The Minimalist Tools for End User Development held by Mary Beth Rosson, and The Semiotics of Visual Programming and Program Visualisation by James Noble and followed by two workshops: Foundations of Spreadsheets (FoS'04), organised by Martin Erwig, and Visual Languages and Formal Methods (VLFM'04) organised by Mark Minas. The conference also held a Doctoral Student Consortium, sponsored by the NSF on the theme of Designing for Diversity in End-User Development Tools.

The main conference received 65 full-length paper submissions, and 16 technical note submissions. The final program comprised 21 full-length papers and 21 technical notes. Including satellite events it attracted about 150 participants.

It follows a more detailed report on one of the satellite workshops.

Visual Languages and Formal Methods

Visual languages and formalisms have always been used in computer science and other fields. With primarily visual methods like UML and SDL, they have become a standard in the process of software analysis, design, and implementation. Moreover, there are many formal modelling and specification approaches where visual approaches are existing or evolving, e.g., for specifying transformations for MDA or for OCL, in order to make such approaches more comprehensible and usable. On the other hand, formal methods are inevitable for specifying, designing as well as implementing visual languages and methods.

The workshop on *Visual Languages and Formal Methods (VLFM'04)* on September 30, 2004 in Rome, Italy, aimed to bring together members of these different areas and to provide a forum for presenting and discussing new results, ideas, and experience among them. The workshop was a satellite event of the IEEE symposium on Visual Languages and Human-Centric Computing (VL/HCC'04) focused on the design, formalization, implementation, and evaluation of computing languages and environments that are easier to learn, easier to use, and easier to understand by a broader group of people.

VLFM'04 followed the line of successful symposia on visual languages and formal methods, which were before held 2001 at Stresa, Italy, and 2003 at Auckland, New Zealand, as symposia within the IEEE symposia on Human Centric Computing HCC'01 and HCC'03, respectively. HCC was the successor of the IEEE symposium on Visual Languages and has now evolved to the IEEE symposium on Visual Languages and Human-Centric Computing (VL/HCC'04).

The workshop program consisted of four sessions. Nine papers were presented in three regular sessions on *Formal Methods Applied to UML*, on *Formal Modelling and Specification Languages with Visual Representations*, and on *Methods and Tools for Processing Visual Language Specifications*. A fourth, more informal session was dedicated to tool presentations.

Mark Minas (Univ. of the Federal Armed Forces, Munich, Germany)

ICGT 2004

ICGT 2004 was the Second International Conference on Graph Transformation following the first one in Barcelona (2002), and a series of six international workshops on graph grammars with applications in computer science between 1978 and 1998. ICGT 2004 was held in Rome (Italy), Sept. 29 - Oct. 1, 2004 under the auspices of the European Association of Theoretical Computer Science (EATCS), the European Association of Software Science and Technology (EASST), and the IFIP WG 1.3, Foundations of Systems Specification.

The conference featured three invited speakers:

- Margaret-Anne Storey on Improving Flow in Software Development through Graphical Representations jointly with VL/HCC,
- Ravi Sandhu on A Perspective on Graphs and Access Control Models,
- Andy Evans on Transformation Language Design: A Metamodelling Foundation,

as well as 26 technical papers selected out of 58 submissions. The topics have ranged over a wide spectrum, including graph theory and graph algorithms, theoretic and semantic aspects, modelling, applications in chemistry and biology, and tool issues.

The satellite events have included two tutorials

- Foundations and Applications of Graph Transformation by Luciano Baresi and Reiko Heckel
- DNA Computing and Graph Transformation Tero Harju, Ion Petre, and Grzegorz Rozenberg

as well as workshops on Graph-based Tools, Petri Nets and Graph Transformations, Software Evolution through Transformations, and Term Graph Rewriting, as well as on Logic, Graph Transformation, Finite and Infinite Structures. Including satellites, the conference attracted about 160 participants.

Below we report on some of the satellite workshops in more detail.

Graph-Based Tools

Graphs are well-known, well-understood, and frequently used means to describe structural aspects in various fields of computer science. They are successfully used as the underlying structuring concept in application domains such as compiler compiler toolkits, constraint solving problems, generation of CASE tools, model-driven software development, pattern recognition techniques, program analysis, software engineering, software evolution, software visualization and animation, and visual languages. In all these domains, tools have been developed which store, retrieve, manipulate and display graphs. It was the purpose of this workshop to summarize the state of the art of graph-based tool development, bring together developers of graph-based tools in different application fields and to encourage new tool development cooperations.

This workshop was of special relevance for the conference on graph transformation: In many cases, the application of graph transformation technology requires the existence of reliable, user-friendly and efficiently working graph transformation tools. These tools in turn have to be built on top of basic

services or frameworks for graphs, the main topic of our workshop. Further important topics of this workshop were common exchange formats for graphs and the application of graph-based tools for visual modelling techniques. In a special session, graph-based tools for the syntax and semantics specification of statecharts have been presented which was of special interest for SegraVis, since new solutions for the definition and implementation of visual modelling languages were presented.

More information about the workshop including its program and an electronic version of all accepted papers appearing in the electronic notes of Theoretical Computer Science (ENTCS) can be found on the workshop web page: `http://tfs.cs.tu-berlin.de/grabats`

Tom Mens (Université de Mons-Hainaut, Belgium)
Andy Schürr (Technical University Darmstadt, Germany)
Gabriele Taentzer (Technical University Berlin, Germany)

Petri Nets and Graph Transformations

Both Petri nets and graph transformation systems are established specification formalisms for concurrent and distributed systems. It is well-known that the token game of place-transition nets can be modeled by double pushouts of discrete labeled graphs. This allows to relate basic notions of place-transition nets like marking, enabling, firing, steps, and step sequences to corresponding notions of graph grammars and to transfer semantical concepts from Petri nets to graph grammars. Since a marking of a net, like a graph in a graph grammar, corresponds to the state of a system to be modeled, graph grammars can be seen to generalize place-transition nets by allowing more complex structured states. The concurrent semantics of graph transformations is strongly influenced by corresponding semantical constructions for Petri nets.

More recently, the modification of Petri nets using graph transformation rules has led to Petri transformations, extending the classical theory by a rule-based technique to study changes in the Petri net structure. This enables the stepwise development of Petri nets.

Other fields of cross-fertilization between Petri nets and graph transformation systems include graph transformations for the simulation or animation of various types of Petri nets, the development of Petri net-like analysis techniques for graph transformation systems, the use of Petri nets to control graph transformation systems, etc.

The workshop brings together people working in the area of low-level and high-level Petri nets and in the area of graph transformation and high-level replacement systems. According to the main idea and in order to further stimulate the research in this important area, this workshop triggers discussion and transfer of concepts across the borders of these and related areas. Eight invited contributions cover a wide range of topics discussed above.

More information can be found at `icgt2004.dsi.uniroma1.it/PNGT.html`.

Grzegorz Rozenberg (Leiden University, The Netherlands)
Hartmut Ehrig (Technical University Berlin, Germany)
Julia Padberg (Technical University Berlin, Germany)

Software Evolution

There are different strategies to address software evolution, i.e., the incremental change of software artifacts driven by feedback from users and other stakeholders or more changes of functional or non-functional requirements.

Model-driven development, as proposed by the OMG's MDA initiative, addresses evolution by automating the transformation of platform-independent models into code, so that evolution can happen at the level of design models. Classical re-engineering technology, instead, starts at the level of programs which, due to the absence or poor quality of models, provide the only definite representation of a legacy system.

To understand which is preferable in each case, and how both approaches could be combined, we require

- a *uniform understanding* of software evolution phenomena at the level of both models and programs, as well as of their interrelation;
- a *common technology basis* that is able to realize the manipulation of artifacts at the different levels, and to build bridges between them.

Graphs, seen as conceptual models as well as data structures, defined by meta models or other means, and *transformations*, given as program or model transformations on graph- or tree-based presentations, provide us with unifying models for both purposes.

Based on this conceptual and technological unification, the workshop addressed the following topics.

- graph-based models for analysis, visualization, and re-engineering
- software refactoring and architectural reconfiguration
- model-driven engineering and model transformations
- consistency management and co-evolution
- relation and tradeoffs between model- and program-based evolution

The workshop was financially supported by the *European Science Foundation (ESF)* through the Scientific Network RELEASE (Research Links to Explore and Advance Software Evolution). It was also the official kick-off meeting of the Working Group on Software Evolution of the *European Research Consortium for Informatics and Mathematics (ERCIM)* and a meeting of the *European Research Training Network* SegraVis on Syntactic and Semantic Integration of Visual Modeling Techniques.

The workshop itself was very successful. There were 30 participants, coming from 11 different European countries (Austria, Belgium, France, Germany, Greece, Italy, Luxemburg, Netherlands, Spain, Switzerland, United Kingdom). More detailed information about the workshop is available online at `http://wwwcs.upb.de/cs/ag-engels/ag_engl/Segravis/Events/SETra04/`

Reiko Heckel (University of Paderborn, Germany)
Tom Mens (Université de Mons-Hainaut, Belgium)

Term Graph Rewriting

TERMGRAPH 2004, the 2nd International Workshop on Term Graph Rewriting, took place in Rome on Saturday 2nd October 2004 as a satellite event of the 2nd International Conference on Graph Transformations (ICGT 2004). The first TERMGRAPH workshop took place in Barcelona in 2002, also as a satellite event of ICGT.

Term graph rewriting is an active domain of research, concerned with the representation of expressions as graphs and their evaluation by rule-based graph transformation. The advantage of representing terms by graphs rather than strings or trees as in standard term rewriting systems is that common subexpressions can be shared, which improves the efficiency of computations in space and time. Efficiency and sharing have been important research topics in the last twenty years, and a wide range of mechanisms have been used for sharing in implementations of programming languages. Many functional, logic, object-oriented and concurrent calculi are based on term graphs. Term graphs are also used in symbolic computation systems and automated theorem proving.

Research in term graph rewriting ranges from theoretical questions to practical implementation issues. Topics of interest include:

- Theory of first-order and higher-order term graph rewriting.
- Graph rewriting in the lambda-calculus and related calculi.
- Applications in functional, logic, object-oriented and concurrent languages.
- Applications in automated reasoning and symbolic computation.
- Implementation issues.

The aim of this workshop is to bring together researchers working in these different areas to foster their interaction, to provide a forum for presenting new ideas and work in progress, and to enable newcomers to learn about current activities in term graph rewriting.

The workshop included sessions on programming language design using graphical structures, graphical encodings of functional and object-oriented programming languages, and applications of term graphs in logic. In addition, there was an invited lecutre on sharing graphs, by Stefano Guerrini (University of Rome, Italy).

The Proceedings of TERMGRAPH 2004 will appear in Elsevier's Electronic Notes in Theoretical Computer Science. More information can be found on the workshop web page at

`www.dcs.kcl.ac.uk/staff/maribel/TERMGRAPH.html`

Maribel Fernandez (King's College London, United Kingdom)

The Role of the Software Architect: The Software Architecture Memorandum of the ”Sylter Runde”

Ralf H. Reussner *
*University of Oldenburg / OFFIS, Germany

Abstract. *On September 23rd and 24th, 2004, a round-table meeting of the ”Sylter Runde” in Westerland, Sylt, was devoted to the role of software architects in Germany. The meeting was organised by Professor Dr. Norbert Szyperski and Professor Dr. Wolfgang König. The group focused on measures to establish the role of the software architect and software architecture as a discipline in Germany. A report, the ”Sylter Memorandum on Software Architecture” was issued. This EASST newsletter article summarises the discussions of the meeting and highlights the the recommendations of the Memorandum.*

Keywords: Software Architect, Memorandum

1 The Meeting

The ”Sylter Runde” is a round-table discussion forum founded and moderated by Professor Dr. Norbert Szyperski. It is held regularly in Westerland, Sylt, Germany and is devoted to in-depth discussions of various topics relevant to modern society.

The reported meeting of the ”Sylter Runde” was concerned with the question whether the introduction of a role of a software architect comparable to the above described role of the building architect can help to overcome problems of software engineering (as those commonly referred to in the context of the so-called software crisis, such as late project schedules or the inability to deal with fast changing requirements in large projects). Related to this role of an architect as an advocate and coordinator, is the question how to establish such a role of software architects by education, best-practices and guidelines for commissioning software projects.

Matured production organisations make use of the benefits of specialisation. This speciaisation refers to differentiated roles for software developers as well as to the existence of specialised components. Their integration into a project (or resp. into a novel product) is planned and controlled during production by a designer and a coordinator. In building architecture, this role of design and coordination is filled by an architect. The analogy software design to building design is not new. In the 1960s Brooks and others coined the term software architecture for high-level software designs. However, the role of the software architect is nowadays primarily seen as chief-designer. This differs significantly from the role of the building architect who certainly is not in the role of the chief mason. Much more a building architect

accomplishes tasks like requirements elicitation, requirements specification, high-level design, quality assurance, etc. Briefly, the architect is the advocate of the customer or the owner. This is emphasised by the existence of independent architects, i.e., architects not being employed by any of the participating construction companies.

2 Summary of the Memorandum

One of the positive aspects of the analogy between building and software architecture is the emphasis on single development projects without a separation of development and production. While for most engineered products one can distinguish a development and a production process, this separation does not exist for buildings and for software. However, in spite of the appealing aspects of the software architecture vs. building architecture analogy, the participants also agreed that the power of the analogy is rather limited, in particular, as software is immaterial. This results in broken metaphors when it comes to software-reuse. In addition, the architecture analogy does not show the importance of the engineering aspects needed to construct software, such as predictable behaviour, etc.

However, the participants agreed that despite all problems of the architecture analogy, the analogy between the *roles* of the building architect and the software architect is in fact very fruitful.

To strengthen the role of the software architect as an advocate of the customer and as an independent profession, the memorandum considers the following actions as indispensable:

- Definition of the skills of software architects. Exemplary, the following skills are considered as relevant for software architects: Sound understanding of corporate business and technological strategies, deep understanding of domain requirements and the technology used, understanding of the business goals, team management, credibility due to competence, etc.
- Clear separation of responsibilities within software projects.
- Mandatory commissioning of a software architect in public software development calls for bids.
- Education of software architects according to adequate curricula. Although a software architect will not get his or her certificate solely on basis of an academic education, we need to create new curricula for software architects, e.g., by application domain-specific software architecture master programmes. In the area of enterprise applications, the curricula of information systems (”Wirtschaftsinformatik”) studies could form a base. For embedded systems, the more technical oriented computer science education might be a good starting point for developing such curricula.
- Fostering the exchange of information and experience among software architects.
- Publication of best-practice examples of successful architectures and projects implementing these architectures.
- Strengthening the separation of labour in software engineering. This implies a higher degree of specialisation and domain-orientation, comparable to the building or manufacturing fields.

- Definition and standardisation of commonly accepted software quality standards and their certification.
- Creation of a centre for software architecture: The Software-Bauhaus.

The Software-Bauhaus is thought to be a centre for software architecture education and practice, following the legendary Bauhaus from Weimar in the twenties of the last century. This Bauhaus is an archetype for a Software-Bauhaus, due to its close relationship between teachers and students, its quest for excellence, and its strict orientation to the applicability and innovation of its products.

The full text of the Memorandum will be available in German at

`http://www.sylter-runde.de`

List of Participants

Gerhard Barth	Frankfurt am Main
Dirk Buschmann	Köln
Ulrich Dietz	Villingen-Schwenningen
Eduard Heindl	Furtwangen
Klaus Höring	Bergisch Gladbach
Stefan Kirn	Hohenheim
Cuno Pfister	Zürich
Ralf Reussner	Oldenburg
Sebastian Saxe	Altenholz
Frank P. Schmitz	Berlin
Johannes Siedersleben	München
Norbert Szyperski	Köln / Westerland
Marcus Wiebcke	Hamburg

Bogor: An Extensible Framework for Domain-Specific Model Checking

John Hatcliff *and Matthew B. Dwyer **and Robby *
*SAnToS Laboratory, Kansas State University
{hatcliff,robby}@cis.ksu.edu
**e^2 Laboratory, University of Nebraska, Lincoln
dwyer@cse.unl.edu

Abstract. *Model checking has proven to be an effective technology for verification and debugging in hardware and more recently in software domains. We believe that recent trends in both the requirements for software systems and the processes by which systems are developed suggest that domain-specific model checking engines may be more effective than general purpose model checking tools. To overcome limitations of existing tools which tend to be monolithic and non-extensible, we have developed an extensible and customizable model checking framework called Bogor. In this article, we summarize how Bogor provides direct support for modeling object-oriented designs and implementations, and how its modeling language and algorithms can be extended and customized to create domain-specific model checking engines.*

Keywords: verification, software model checking, domain-specific modeling

1 Trends that motivate domain-specific model checking

Temporal logic model checking [CGP00] is a powerful framework for reasoning about the behavior of finite-state system descriptions and it has been applied, in various forms, to reasoning about a wide-variety of software artifacts. For example, model checking frameworks have been applied to reason about software process models, (e.g., [KGMW00]), different families of software requirements models (e.g., [CAB$^+$01]), architectural frameworks (e.g., [GKK03]), design models, (e.g., [HDD$^+$03]), and system implementations (e.g., [BR00, BHPV00, CDH$^+$00, God97]). The potential of model checking technology for: (a) detecting coding flaws that are hard to detect using existing quality assurance methods (e.g., such as bugs that arise from unanticipated interleavings in concurrent programs), and (b) verifying that system models and implementations satisfy crucial temporal properties and other light-weight specifications has led a number of international corporations and government research labs such as Microsoft, IBM, Lucent, NEC, NASA, and Jet Propulsion Laboratories (JPL) to fund their own software model checking projects.

The effectiveness of the software model checking applications cited above has in most cases relied on

the fact that the researchers carrying out the studies had a detailed knowledge of the model checking tool being applied and understood very clearly (perhaps as the result of extensive experimentation) how to (a) map the problem being modeled to the tool in a manner that would draw on the strengths of the tool, and (b) how to configure the tool for optimal performance. In some cases, researchers were not satisfied with the results of mapping their problem to an available configuration of an existing tool, and they found it necessary to study an *existing* model checking framework in detail in order to customize it [CAB+01, DIS99]. In other cases, researchers had the insight that they needed to develop a *new* framework targeted to the semantics of a family of artifacts (e.g., Java programs [BHPV00]) or a specific domain (e.g., device drivers [BR00]).

We believe that are a number of trends both in the requirements of computing systems and in the processes by which they are developed that will drive persons and organizations interested in applying model checking technology to rely increasingly on customization/adaptation of existing tool frameworks or construction of new model checking tools.

- **Model-driven development:** Increasing emphasis on using a rich collection of interlinked models at a variety of levels of abstraction for not only software design but also software implementation will drive researchers to look beyond model checkers with fixed input languages that target a single level of abstraction. Current technology allows model checking to be applied at the source code and byte code (or machine code level) as well as higher level models – but for the most part, different tools with different characteristics are used for these different levels of abstraction. We envision a single integrated framework in which artifacts at all levels of abstraction are checkable, in which relationships such as refinement/conformance are established by model checking collections of artifacts at different levels of abstraction, and in which verification models are formed by composing: (a) analyzable models from which code is automatically generated, (b) manually written source that was not amenable to auto-generation, and (c) high-level specifications of cross-cutting *aspects* such as synchronization or event delivery policies.

- **Product-line architectures and reusable middleware infrastructure:** Large-scale distributed systems are increasingly built on top of middleware frameworks such as CORBA and .NET and use software product-line architectures that encourage the reuse of software components and infrastructure across applications. The size (often 100's of thousands of lines of code) and the complexity of this infrastructure (which often contains extensive use of object-oriented design patterns and complex heap structures) will make it difficult to apply existing techniques for model-extraction from source code, but the high degree of reuse of this infrastructure will make it feasible to construct custom-built verification model components and checking engines that are tailored to those frameworks. Moreover, as greater automation is sought in development and validation processes, developers will have a greater difficulty applying general purpose model checking tools which require a higher degree of configuration and interaction, and product-line architects will increasingly seek to reduce developer involvement by building model checking infrastructure that is dedicated to a particular software architecture, domain, and development process.

- **Sheer variety of application domains and computation models:** As software is embedded in an ever broadening range of devices and as software increases in scale, verification experts will

increasingly move away from general purpose solutions as they seek to achieve scalability by developing search algorithms, state-storage approaches, and model reduction strategies that leverage properties of a particular domain. For example, we have observed that reduction methods designed for a particular application domain (e.g., for example, Java programs) may be ineffective for a different application domain such as avionics systems where timing issues play a greater role. Conversely, we have found that the specialized quasi-cyclic structure of computation in a certain class of avionics systems enables dramatic optimizations in state space search and state storage strategies that are customized for this particular domain [DDH$^+$02, DRDH03].

Thus, there is a need for model checking tools that support customization, extensibility, and that are easily embedded or encapsulated in larger development tools. Existing model checkers, however, including widely used tools such as SPIN [Hol97], FDR2 [For03], and NuSMV [CCGR00], are designed to support a fixed input language using a fixed collection of state-space representations, reduction and exploration algorithms. The capabilities of these tools has evolved over time, but that evolution has been limited to the capabilities that the tool developer themselves found useful or desirable. Moreover, most existing tools do not provide direct support for features found in object-oriented software systems. Recent versions of the SPIN model checker allow one to include C source directly in the Promela modeling language. While this does allow for a degree of extensibility, one might how for more comprehensive mechanisms for extensibility that are part of the overall design of the model checking framework.

While it is possible to continue the practice of cannibalizing and modifying existing model checkers, or building new model checkers from scratch, the level of knowledge and effort required for such activitivies currently prevents many *domain* experts who are not necessarily experts in model checking from successfully applying model checking to software analysis. Even though experts in different areas of software engineering have significant domain knowledge about the semantic primitives and properties of families of artifacts that could be brought to bear to produce cost-effective semantic reasoning via model checking, in order to make effective use of model checking technology these experts should not be required to build their own model checker or to pour over the details of an existing model checker implementation while carrying out substantial modifications.

2 Bogor: a customizable and extensible framework

To meet the challenges of using model checking in the context of current trends in software development outlined above, we have constructed an extensible and highly modular explicit-state model checking framework called Bogor [RDH03, Lab03a]. Using Bogor, we seek to enable more effective incorporation of domain knowledge into verification models and associated model checking algorithms and optimizations, by focusing on the following principles.

- **Direct Support of Object-Oriented Languages:** Bogor provides a rich base modeling language including features that allow for dynamic creation of objects and threads, garbage collection, virtual method calls and exception handling. For these primitives, we have extended Bogor's default algorithms to support state-of-the-art model reduction/optimization techniques that we have developed [DHRR04, RDHI03] by customizing for object-oriented software existing techniques such

as collapse compression [Hol97], heap symmetry [Ios02], thread symmetry [BDH02], and partial-order reductions.

- **Extensible Modeling Language:** Bogor's modeling language can be extended with new primitive types, expressions, and commands associated with a particular domain (e.g, multi-agent systems, avionics, security protocols, etc.) and a particular level of abstraction (e.g., design models, source code, byte code, etc.)
- **Open Modular Architecture:** Bogor's well-organized module facility allows new algorithms (e.g., for state-space exploration, state storage, etc) and new optimizations (e.g., heuristic search strategies, domain-specific scheduling, etc.) to be easily swapped in to replace Bogor's default model checking algorithms.
- **Design for Encapsulation:** Bogor is written in Java and comes wrapped as a plug-in for *Eclipse* – an open source and extensible universal tool platform from IBM. This allows Bogor to be deployed as a stand-alone tool with a rich graphical user interface and a variety of visualization facilities, or encapsulated within other development or verification tools for a specific domain.
- **Pedagogical Materials:** Even with a tool like Bogor that is designed for extensibility, creating customizations requires a significant amount of knowledge about the internal Bogor architecture. To communicate this knowledge, we have developed an extensive collection of tutorial materials and examples. Moreover, we believe that Bogor is an excellent pedagogical vehicle for teaching foundations and applications of model checking because it allows students to see clean implementations of basic model checking algorithms and to easily enhance and extend these algorithms in course projects. Accordingly, we have developed a comprehensive collection of course materials [Lab03b] that have already been used in graduate level courses on model checking at several institutions.

In short, Bogor aims to be not only a robust and feature-rich software model checking tool that handles the language constructs found in modern large-scale software system designs and implementations, it also aims to be a model checking *framework* that enables researchers and engineers to create families of domain-specific model checking engines.

3 Bogor's support for object-oriented language features

Bogor checks systems specified in a revised version of the Bandera Intermediate Representation (BIR) [IDH04]. The previous version of BIR was designed to be an intermediate language used by Bandera for translating Java programs to the input languages of existing model checkers such as Spin. Thus, this earlier version provided direct support for modeling Java features such as threads, Java locks (supporting wait/notify), and a bounded form of heap allocation. To facilitate the construction of translators to back-end model checkers like SPIN, BIR control-flow and actions are stated in a *guarded command* format which it quite close to the format used to specify systems in model checker input languages like Promela.

Our experience with Bandera and other tools such as JPF and dSpin has led us to conclude that software model checking can be more effectively supported by a new infrastructure that has at its core an extensible

```
extension Set for myPackage.SetModule {
    typedef type<'a>; // declare a type Set.type

    expdef Set.type<'a> create<'a>('a ...); // create a set with zero or more elements
    expdef 'a chooseElement<'a>(Set.type<'a>); // non-deterministically choose an element
    expdef boolean isEmpty<'a>(Set.type<'a>); // check for emptiness
    expdef boolean forAll<'a>('a -> boolean, Set.type<'a>); // check predicate over all elements

    actiondef add<'a>(Set.type<'a>, 'a); // add an element to a set
    actiondef remove<'a>(Set.type<'a>, 'a); // remove an element from a set
}
```

Figure 1: BIR Set Extension Declaration (excerpt)

model checker that is designed to support software directly rather than relying on translations to model checkers that do not provide direct support for modeling many of the language features found in modern software. As part of this transition to a new infrastructure, we have revised the definition of BIR to include a number of new features such as the BIR extension mechanism, generic types and polymorphic functions, type-safe function pointers, virtual function/method tables, and exceptions.

In contrast to the input languages used in almost all other model checkers (including SPIN [Hol97], NuSMV [CCGR00], SLAM [BMMR01], BLAST [HJMS02]), BIR supports unbounded dynamic creation of both thread and heap objects with automatic reclamation by garbage collection. Moreover, BIR provides a state-of-the-art canonical heap representation that seems essential for effective checking of highly dynamic concurrent software systems. Such systems generate many heap instances that differ in the relative position of allocated objects but that are actually *observationally equivalent* (i.e., the heap instances can not be distinguished by any Java memory operations). Due to positional differences of object placement in heaps, conventional representations of heaps as, for example, arrays of memory cells would yield different states for these heaps. However, Bogor's canonical heap representation (based on work by Iosif on dSpin [DIS99]) ensures that heaps that are observationally equivalent map to the same state. This dramatically reduces the number of states generated when checking highly dynamic systems.

4 Bogor's extensible modeling language

BIR's extension facility allows modelers to add new types, expressions, and commands. Figure 1 presents a set extension declaration. An extension declaration consists of (1) a signature declaration which specifies the new symbols and associated arities to be introduced into the name-spaces for types, expressions, and actions, and (2) the name of a Java package that implements the semantics of the extension. Note that the extension does not extend the BIR grammar, but only adds names to the set of names of built-in expressions, actions, etc. This means that the developer does not need to extend the parser or other syntactic support facilities. Rather, the developer traverses the extension structure and implements the extension semantics using well-defined APIs for BIR's existing abstract syntax trees and Bogor model checker components.

Extensions provide a convenient way to realize domain-specific abstractions of software components

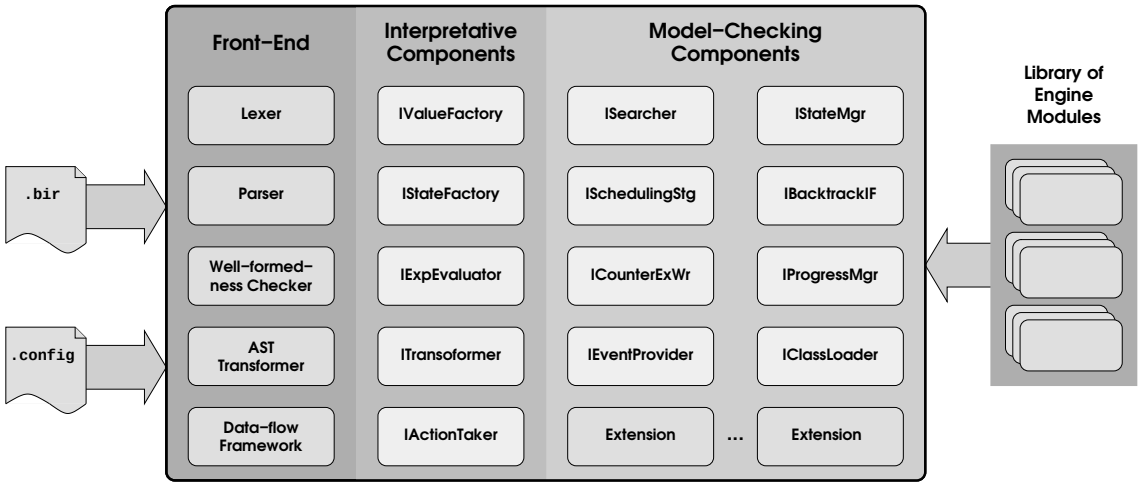


Figure 2: Architecture and primary modules of Bogor

or layers, and to address the input language gap for modeling domain-specific software artifacts. Often times, there are component/layers of the software that have a significant amount of state that is *irrelevant* to the properties being checked. Rather than maintaining a complete implementation of a software component/layer using BIR's variables, the Java package implementing the extension can hold the state associated with the complex component/layer and only expose as much as is relevant at the BIR level. Since only the BIR variables are held in Bogor's state vector during model checking, this can dramatically reduce the costs of representing portions of a software system.

Hiding complex portions of the state in this manner is not novel. For example, when modeling communication protocols using Spin [Hol97], channel data types (`chan`) are often used to model message-passing channel in networks. This can be done because the specific implementation of the network channels is *irrelevant* with respect to the properties being checked. The properties are usually only concerned with the functional behavior of the channels, i.e., rendezvous, asynchronous, synchronous, and sending and receiving messages. Furthermore, properties are usually only concerned with a channel's *abstract* states, i.e., the specific channel implementation state, for example, the state of message retransmission protocols used to provide the channel service, does not need to be exposed in the model state. What is novel about BIR is that it does not a priori hard-code such abstraction mechanisms for a particular domain – rather, BIR's extension facility provides an open-ended mechanism for adding any number of domain-specific abstractions. For example, BIR extensions are used to represent the functionality of the Real-Time CORBA Event Service in our work on checking CCM designs [DDH+02].

5 Bogor's open architecture

Figure 5 presents the Bogor architecture. The architecture of Bogor can be divided into three parts: (1) a front-end that parses and type checks a given model expressed in the BIR modeling language, and (2) interpretive components that implements the values the state transformations implied by BIR's semantics,

and (3) model checking engine components that implement search and state storage strategies.

All model checking tools include functional aspects for state-space search, scheduling, and managing seen before states. However, in their implementations, these aspects are often tangled, and thus insertion of alternate strategies or other customizations is often quite difficult. One of the contributions of our work is not just to present a non-tangled implementation, but moreover to present core components using widely-used and well-documented design patterns [GHJV95] that hide irrelevant implementation details by encapsulation, that reduce dependences between components, and that build in strategies for parameterization, adaptation, and extension.

For example, the `ISearcher` (implemented using the STRATEGY pattern) defines the search method used. For example, if depth-first search is used, then at any given state, its children states will be explored first before exploring its sibling states. The `IStateManager` (implemented using the FACADE pattern) for provides a interface for storing states and for determining whether or not a state has been visited before. The `ISchedulingStrategist` (STRATEGY pattern) determines the scheduling strategy employed by the model checker. The most common strategy of model checkers is to generate all possible interleavings of thread executions. Other strategies include incorporation of support for priority based scheduling. In addition, when processing any inner-thread non-deterministic choice (e.g., associated with multiply-enabled transitions within the same thread), this module should be consulted to determine which transition to execute next. For example, in a full-state exploration mode, the scheduler should make sure that every branch of a non-deterministic choice should be explored. This module is also consulted to determine which transformations are enabled in a given state.

More details about Bogor modules can be found in [RDH03] and a complete listing of module APIs and examples of their use can be found on the Bogor web-site [Lab03a].

6 Bogor's design for encapsulation

Bogor is implemented as a plug-in for *Eclipse* – a freely available open source and extensible universal tool platform from IBM. Eclipse provides a rich set of infrastructure for, for example, creating integrated development environments (IDEs), graphical editors, etc., that is ideal for building a user interface (UI) for a model checking tool. Eclipse provides a plugin facility by which one can add more functionality. The plugin facility of Eclipse matches well with Bogor's module extension facility. Eclipse, like Bogor, is implemented in Java. Thus, it allows a tight integration of Bogor and its GUI context and extensions.

Our own experience of customizing Bogor with domain-specific modeling primitives and optimizations and encapsulating the resulting customized tool within larger environments has been quite positive.

We are developing the next generation of the Bandera [CDH$^+$00] tools in Eclipse. This new version of Bandera analyzing models generated from Java source code using Bogor's rich built-in support for object-oriented language features. For these primitives, we have extended Bogor's default algorithms to support state-of-the-art model reduction/optimization techniques that we have developed by customizing for object-oriented software existing techniques such as collapse compression, heap symmetry, thread symmetry, and partial-order reductions. [DHRR04, RDHI03].

In our work on the Cadena development environment (also built in Eclipse) [HDD$^+$03] for designing large-scale distributed real-time embedded systems built using the CORBA Component Model (CCM),

we have extended Bogor's modeling language to include APIs associated with the CORBA component model and an underlying real-time CORBA event service. [DDH+02, DRDH03]. For checking avionics system designs in Cadena, we have customized Bogor's scheduling strategy to reflect the scheduling strategy of the real-time CORBA event channel, and created a customized parallel state-space exploration algorithm that takes advantage of properties of periodic processing in avionics systems. These customizations for Bandera and Cadena have resulted in space and time improvements of over three orders of magnitude compared to our earlier approach [CDH+00, HDD+03] which created models for Spin and dSpin.

In other work, we have extended Bogor to support checking of specifications written in the Java Modeling Language (JML) [RRDH04, Lab04] and GUI frameworks [DRTV04]. Researchers outside of our group have extended Bogor to support checking of programs using AspectJ.

7 Pedagogical support for Bogor

The success and the expediency of the customization efforts described above were determined by several factors: (1) accessibility to domain knowledge, (2) intimate understanding of existing model checking algorithms, and (3) a model checking framework that allowed us to rapidly prototype ideas as concrete algorithms that we could experiment with. We believe that these factors influenced not only our specific experiences, but the experiences of others in applying Bogor as well. While issues in (1) should be addressed by the practitioners themselves, it is crucial for us to provide tutorial and reference material about Bogor's architecture and algorithms to enable others to successfully customize Bogor. Moreover, we believe Bogor itself is an excellent pedagogical vehicle for teaching foundations and applications of model checking because it allows students to see clean implementations of basic model checking algorithms and to easily enhance and extend these algorithms in course projects to include a variety of enhancements and optimizations.

Bogor comes with a user manual that includes BIR documentation (e.g., grammar, language description, abstract syntax tree implementation in Java, etc.) and Bogor extension tutorials that are directly accessible through the Eclipse help system as well as in PDF and HTML format. These materials refer to a well-documented repository of examples that illustrate how to construct various types of Bogor extensions.

To use Bogor as a pedagogical vehicle for teaching foundations and applications of model checking, we designed an extensive collection of freely available course materials for a one-semester course `http://model-checking.courses.projects.cis.ksu.edu`.

This course emphasizes a practical and project-oriented approach to learning the technical foundations of model checking and methodologies for applying model checking tools to realistic systems. Foundational topics covered include basic explicit-state reachability algorithms, temporal specification formalisms including LTL and CTL, partial order reductions, state-space representations (collapse compression, etc.), and alternate search strategies. In an approach similar to that used in compiler courses, these foundational and theoretical concepts are reinforced by having students implement key components of an explicit state model checker.

Students learn to apply Bogor to model and analyze simple concurrent systems that illustrate basic

concepts of state-space exploration. Programming projects involve (re)implementing or modifying the core modules of Bogor's model checking engine, or implementing new modeling language primitives using Bogor's extensible modeling language. In addition to simply reinforcing the central concepts of model checking, the overall goal of these implementation exercises is to move students to the point where they can effectively develop model checking tools and associated methodologies for verification of real world systems by tailoring Bogor to different application domains.

Methodological aspects of model checking (and Bogor, in particular) are also emphasized. This includes repeatable strategies for capturing concurrent/distributed systems as effective verification models, applying abstraction and other state-space reducing model transformations, and using a pattern-based approach to constructing temporal specifications.

The course distribution for instructors includes a variety of pedagogical materials such as typeset lecture notes and guided exercises, PowerPoint lecture slides, streaming video for our lectures, source code for lecture examples, weekly quizzes and solutions, homeworks and solutions, exams and solutions. A separate distribution for students includes only the lecture slides and examples.

8 Conclusion

In summary, we believe that a number of trends in software development suggest the need for flexible and adaptable infrastructure to enable practitioners to more effectively develop model checking tools that are customized to particular domains and development processes. While there are many avenues that one might pursue, we believe that the Bogor framework provides a robust and well-reasoned foundation that researchers and practitioners can use to address important facets of automated reasoning for modern software systems.

References

[BDH02] Dragan Bosnacki, Dennis Dams, and Leszek Holenderski. Symmetric SPIN. *International Journal on Software Tools for Technology Transfer*, 4(1):92–106, 2002.

[BHPV00] Guillaume Brat, Klaus Havelund, SeungJoon Park, and Willem Visser. Java PathFinder – a second generation of a Java model-checker. In *Proceedings of the Workshop on Advances in Verification*, July 2000.

[BMMR01] Thomas Ball, Rupak Majumdar, Todd Millstein, and Sriram Rajamani. Automatic predicate abstraction of C programs. In *Proceedings of the ACM SIGPLAN '01 Conference on Programming Language Design and Implementation (PLDI-01)*, pages 203–213, June 2001.

[BR00] Thomas Ball and Sriram Rajamani. Bebop: a symbolic model-checker for boolean programs. In K. Havelund, editor, *Proceedings of Seventh International SPIN Workshop*, volume 1885 of *Lecture Notes in Computer Science*, pages 113–130. Springer-Verlag, 2000.

[CAB+01] William Chan, Richard J. Anderson, Paul Beame, David Notkin, David H. Jones, and William E. Warner. Optimizing symbolic model checking for statecharts. *IEEE Transactions on Software Engineering*, 27(2):170–190, 2001.

[CCGR00] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV : a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, 2000.

[CDH+00] James C. Corbett, Matthew B. Dwyer, John Hatcliff, Shawn Laubach, Corina S. Păsăreanu, Robby, and Hongjun Zheng. Bandera : Extracting fi nite-state models from Java source code. In *Proceedings of the 22nd International Conference on Software Engineering*, pages 439–448, June 2000.

[CGP00] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 2000.

[DDH+02] William Deng, Matthew Dwyer, John Hatcliff, Georg Jung, and Robby. Model-checking middleware-based event-driven real-time embedded software. In *Proceedings of the 1st Internatiuonal Symposium on Formal Methods for Component and Objects*, pages 154–181, 2002.

[DHRR04] Matthew B. Dwyer, John Hatcliff, Robby, and Venkatesh R.Prasad. Exploiting object escape and locking information in partial order reduction for concurrent object-oriented programs. *Formal Methods in System Design*, 25(2-3):199–240, 2004.

[DIS99] Claudio Demartini, Radu Iosif, and Riccardo Sisto. dSPIN : A dynamic extension of SPIN. In *Theoretical and Applied Aspects of SPIN Model Checking (LNCS 1680)*, September 1999.

[DRDH03] Matthew B. Dwyer, Robby, Xianghua Deng, and John Hatcliff. Space reductions for model checking quasi-cyclic systems. In *Proceedings of the Third International Conference on Embedded Software*, 2003.

[DRTV04] Matthew B. Dwyer, Robby, Oksana Tkachuk, and Willem Visser. Analyzing interaction orderings with model checking. In *Proceedings of the 19th IEEE Conference on Automated Software Engineering*, 2004. (to appear).

[For03] FormalSystems. FDR2 website. `http://www.fsel.com/`, 2003.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley Pub. Co., 1995.

[GKK03] David Garlan, S. Khersonsky, and Jung Soo Kim. Model checking publish-subscribe systems. In *Proceedings of the 10th International SPIN Workshop on Model Checking of Software*, pages 166–180, May 2003.

[God97] Patrice Godefroid. Model-checking for programming languages using VeriSoft. In *Proceedings of the 24th ACM Symposium on Principles of Programming Languages (POPL'97)*, pages 174–186, January 1997.

[HDD+03] John Hatcliff, William Deng, Matthew Dwyer, Georg Jung, and Venkatesh Prasad. Cadena: An integrated development, analysis, and verifi cation environment for component-based systems. In *Proceedings of the 25th International Conference on Software Engineering*, pages 160–173, 2003.

[HJMS02] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Gregoire Sutre. Lazy abstraction. In *Proceedings of the 29th ACM Symposium on Principles of Programming Languages*, pages 58–70, January 2002.

[Hol97] Gerard J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–294, May 1997.

[IDH04] Radu Iosif, Matthew B. Dwyer, and John Hatcliff. Translating Java for multiple model checkers: the Bandera back end. *International Journal on Formal Methods in System Design*, 2004. (to appear).

[Ios02] Radu Iosif. Symmetry reduction criteria for software model checking. In *Proceedings of Ninth International SPIN Workshop*, volume 2318 of *Lecture Notes in Computer Science*, pages 22–41. Springer-Verlag, April 2002.

[KGMW00] Christos T. Karamanolis, Dimitra Giannakopolou, Jeff Magee, and Stuart M. Wheather. Model checking of workflow schemas. In *4th International Enterprise Distributed Object Computing Conference*, pages 170–181, September 2000.

[Lab03a] SAnToS Laboratory. Bogor Website. `http://bogor.projects.cis.ksu.edu`, 2003.

[Lab03b] SAnToS Laboratory. Software Model Checking course materials website. `http://model-checking.courses.projects.cis.ksu.edu`, 2003.

[Lab04] SAnToS Laboratory. JML-Eclipse Website. `http://jmleclipse.projects.cis.ksu.edu`, 2004.

[RDH03] Robby, Matthew B. Dwyer, and John Hatcliff. Bogor: An extensible and highly-modular model checking framework. In *Proceedings of the 9th European Software Engineering Conference held jointly with the 11th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 267–276, 2003.

[RDHI03] Robby, Matthew B. Dwyer, John Hatcliff, and Radu Iosif. Space-reduction strategies for model checking dynamic software. In *Proceedings of the 2nd Workshop on Software Model Chekcing*, volume 89(3) of *Electronic Notes in Theoritical Computer Science*, 2003.

[RRDH04] Robby, Edwin Rodríguez, Matthew B. Dwyer, and John Hatcliff. Checking strong specifications using an extensible software model checking framework. In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2988 of *Lecture Notes in Computer Science*, pages 404–420, March 2004.

FMICS 2004
Ninth International Workshop on
Formal Methods for Industrial Critical Systems

Juan Bicarregui, Andrew Butterfield, Alvaro Arenas

The Ninth International Workshop on Formal Methods for Industrial Critical Systems (FMICS 04) was held in Linz, Austria, during September 20-21, as a co-located event of the 19th IEEE Conference on Automated Software Engineering,. The workshop series promotes the use of formal methods for industrial applications by supporting research in this area and by serving as a forum for the exchange of ideas between researchers and practitioners, in both industry and academia. This workshop, organised by CCLRC Rutherford Appleton Laboratory, was attended by 35 participants from academia and industry from 16 countries.

The two keynote speakers gave interesting and stimulating presentations. Jeremy Dick from Telelogic spoke on linking formal methods to formal requirements describing how existing tools for supporting requirements traceability could be adapted to work with formal specification and refinement documents. Cedric Fournet of Microsoft Research spoke on the verification of the security of XML-based web-services and described how the “applied” pi-calculus was used to analyse the safety of security policies, work which has contributed to recent revisions of Microsoft code. FMICS would like to thank both invited speakers for their relevant and highly informative contributions to the success of the workshop.

Seventeen submitted papers were presented with authors from 17 countries spanning formal methodologies as diverse as state-charts, model-checking, mixed intuitionistic logic and Boolean equation systems; and applications ranging from operating systems, network services, communications protocols and middleware behaviour, to flight guidance.

The best paper award supported by the European Association of Software Science and Technology (EASST) was awarded to Martin Fränzle and Christian Herde for their paper on proof engines for bounded model checking of hybrid systems.

Other papers presented included Object Oriented concepts identification from formal B specifications; an Abstract Interpretation Toolkit for μCRL; Early Verification and Validation of Critical Systems; and Model Checking Flight Guidance Systems: from Synchrony to Asynchrony, among others.

The proceedings of the workshop are published as a technical reports of the Johannes Kepler University and will appear in Electronic Notes in Theoretical Computer Science. Selected papers will be invited for publication in an special issue of Formal Methods in System Design.

The organisers wish to thank FME and the i-Trust Working Group for sponsorship for the invited speakers. Participants enjoyed a good Austrian dinner courtesy of ERCIM.

Report on ICGT 2004
2nd International Conference on Graph Transformation

Hartmut Ehrig *
*Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin, Germany
Email: ehrig@cs.tu-berlin.de

ICGT is the 2nd International Conference on Graph Transformation following up a series of six international workshops on this topic held from 1978 to 1998 in Europe and the USA, and the 1st International Conference ICGT 2002, held in Barcelona. The conference took place under the auspices of EATCS, EASST, and IFIP WG 1.3. It was organized by Paolo Bottoni (Local Organizing Chair) and Francesco Parisi Presicce from the University of Rome "La Sapienza" and Marta Simeoni (Publicity Chair) with support by the Steering Committee of ICGT, where Gregor Engels and Francesco Parisi Presicce served as chairs of the Program Committee.

ICGT 2004 has been colocated with the IEEE Symposium on Visual Languages/Human Centric Computing and was opened on Wednesday, September 29, with a joint invited lecture of ICGT and VL/HCC "Improving Flow in Software Development through Graphical Representations" by Margaret Anne Storey (University of Victoria, Canada). The joint day of both conferences was finished with a joint panel on the topic "Visual Languages: An Application Field for Graph Transformations or more?" chaired by G. Engels and a joint social event, a boat party on the river Tiber.

The program of the accepted papers for ICGT started on Wednesday with sessions on "Integration Technology" and "Chemistry and Biology" and was continued on Thursday with sessions on "Graph Transformation Concepts", "DPO Theory for High-level Structures", "Analysis and Testing", "Graph Theory and Algorithms", and "Application Conditions and Logic". The second invited lecture was presented by by Ravi Sandhu (George Mason University, USA) and the topic "A Perspective on Graphs and Access Control Models".

The conference dinner on Thursday evening was served at the Forum Hotel with an excellent view to Forum Romanum, one of the most interesting historical sites in Rome. The scientific highlight of the dinner was the announcement of the EATCS-and EASST-award winners for the best theoretical resp. best Software Engineering paper of ICGT.

The EATCS award was given to the paper *Local Computations in Graphs: The case of Cellular Edge Local Computations* by J. Chalopin, Y. Metivier and W. Zielonka.

The EASST award was handed over by the EASST vice president to Bernhard Wesfechtel for the paper *Rule Execution in Graph-Based Incremental Interactive Integration Tools* by S. M. Becker, S. Lohmann and B. Westfechtel.

On Friday morning the ICGT program was interrupted by the official mid-term review of the TMR network SEGRAVIS, where most partners of SEGRAVIS are also actively involved in ICGT. In the afternoon the ICGT conference was continued with sessions on “Transformation of Special Structures” and “Object-Orientation”. The final highlight was the third invited lecture by Andy Evans (University of York, UK) with the title “Transformation Language Design: A Metamodelling Foundation”.

In the closing session the steering committee chairman summarized the highlights of the conference and announced an important new workshop on “Graph- and Model Transformation”, which will be organized by Gabi Taentzer and Gabor Karsai probably in September 2005. Finally the 3rd International Conference on Graph Transformation, ICGT 2006, was announced by Leila Ribeiro to be held in Natal, Brazil. According to the pictures she showed Natal is an excellent place not only for ICGT, but also for recreation in a tropical resort place at the ocean immediately south of the equator. In addition to the main ICGT conference there were two interesting tutorials on Tuesday on “DNA Computing and Graph Transformation’ and “Foundations and Applications of Graph Transformation” and several workshops on Friday and Saturday: The workshops on “Logic, Graph Transformations, Finite and Infinite Structures”, “Petri Nets and Graph Transformation”, “Software Evolution through Transformations: Model-based versus Implementation-Level Solutions”, and “Term Graph Rewriting” attracted again a lage number of participants. The total number of participants of the main conference and the satellite events reached about 120. Taking into account also the high quality of the ICGT papers published in Springer LNCS 3256 (2004) and the contributions of the satellite events, where selected papers will be published in ENTCS, the conference was another great success in the area of graph transformations.

TAPSOFT 2005
Distinguished Lectures

Theory and Practice of Software Development
Past and Future

Berlin, February 17/18, 2005

Announcement and Call for Participation

The "1st International Conference on Theory and Practice of Software Development" was held at TU Berlin in 1985, bringing together leading international experts to discuss future trends of software development. 20 years later – in 2005 – we would like to analyze how far the expectations of 1985 have been fulfilled and what kind of future trends for the period 2005 – 2025 can be expected.

TAPSOFT 2005 is a Distinguished Lecture Series where again international experts from theory and practice will present their views. Based on their own personal experience and their expectations they will create a forum for discussion on the state of the art of software development, its past, its future and the interaction of both.

TAPSOFT 2005 is an event in the tradition of the TAPSOFT conferences from 1985-1997. The TAPSOFT together with the ESOP and TACAS conferences were joined in 1998 to ETAPS, the European Joint Conferences on Theory and Practice of Software. The Distinguished Lectures of TAPSOFT 2005 take place in addition to the ETAPS conference.

Lecture Notes

Abstracts and Position Statements of the lectures of the Distinguished Lecture Series will first be made available in a TAPSOFT 2005 report and later be published in the EATCS Bulletin and the EASST Newsletter.

Organization and Moderation Hartmut Ehrig and Bernd Mahr

Location TU Berlin, Main Building, H 3005
Strasse des 17. Juni 135
10623 Berlin (Charlottenburg)

Time Feb 17, 2005, 12:45 – Feb 18, 15:00

Preliminary Program

Thursday, February 17, 2004

12:45 Opening and Welcome

13:00 **Hartmut Ehrig**
Theory and Practice of Software Development: The Past

13:30 **Ugo Montanari**
Models and Languages for Service Oriented Computing

14:15 Coffee Break

14:30 **David Harel**
Programming Directly from Requirements

15:15 **Peter Pepper**
New Software Paradigms Need New Language Paradigms

16:00 Coffee Break

16:30 **N.N.**
title

17:15 **Günter Hommel**
Open Problems in Modeling Using Stochastic Petri Nets

Friday, February 18, 2005

9:00 **Tiziana Margaria**
Modeling in the context of legacy systems

9:45 **Stefan Jähnichen**
Brain Computer Interfaces -- A Challenge for Systems Engineering?

10:30 Coffee Break

11:00 **Gregor Engels**
Dynamic Binding of Web Services

11:45 **Herbert Weber**
From Programme Engineering to Software Engineering

12:30 Coffee Break

12:45 **Fernando Orejas**
General methods for the semantics of modelling and specification formalisms

13:30 **Bernd Mahr**
Theory and Practice of Software Development: The Future

14:00 Closing and Reception

Registration (free of charge)

Early Registration Deadline: January 10, 2005 with service for hotel booking upon request

Late Registration Deadline: February 1, 2005

Registration after this date cannot be guaranteed. It is generally expected that all registered persons will participate on both days. Otherwise organizers ask for notification as soon as possible. Registration is requested by e-mail to moswald@cs.tu-berlin.de.

Organizers and Further Information

Prof. Dr. Hartmut Ehrig
Tel: +49-30-314-73511
E-mail: ehrig@cs.tu-berlin.de

Prof. Dr. Bernd Mahr
Tel: +49-30-314-73541
E-mail: mahr@cs.tu-berlin.de

Maria Oswald (Secretary)
Tel: +49-30-314-24166
E-mail: moswald@cs.tu-berlin.de

Diana Maaß (Secretary)
Tel: +49-30-314-73540
E-mail: diana@cs.tu-berlin.de

2nd International Workshop on Formal Foundations of Embedded Software and Component-Based Software Architectures (FESCA 2005)

http://www.dcs.kcl.ac.uk/events/fesca/

Satellite event of ETAPS 2005, Edinburgh, Scotland, April 9, 2005,

Scope

The aim of this ETAPS workshop is to bring together researchers from academia and industry interested in formal modeling approaches as well as associated analysis and reasoning techniques with practical benefits for embedded software and component-based software engineering.

Component-based software design has received considerable attention in industry and academia since object oriented software development approaches became popular. Recent years have seen the emergence of formal and informal techniques and technologies for the specification and implementation of component-based software architectures. With the growing need for safety-critical embedded software, this trend has been amplified. Various component models, models of computation and component composition techniques and frameworks have been proposed in the literature.

An underlying theme of all these is to establish correctness or correct by construction software engineering. It is widely believed that formal methods are a good means to achieve correctness in complex software systems. However, advances in formal methods and formal verification have not kept up with the increasing complexity of software. In fact, software complexity is increasing at a pace greater than Moore's law due to the use of software in many facets of computation such as multi-media, real-time systems, networking protocol stacks, complex/real-time operating systems, networked systems software, pervasive networking, etc.

Ubiquitous computing is envisioned as the nature of computation in the near future as evidenced in the emergence of various middleware enabled ubiquitous systems software, such as JINI, .NET, CORBA etc. A quantum leap in the formal methodology and tools is needed to cope with this rising software complexity, or else, correctness guarantees will remain forever elusive. Component software technology gives us some hope because of its inherent 'divide and conquer' nature. However, complex interactions between components pose greater challenges as well. Middleware platforms have proposed a variety of component models (such as OMG's CORBA or implementations of Sun Microsystems's EJB or Microsoft's

.NET). Similarly, meta-models have been developed and standardized to specify component-based software architectures (e.g. UML2.0 or the UML Profile for Enterprise Distributed Object Computing). Also, many embedded software industries (e.g. in avionics) have either in-house or application specific design flows or middleware (e.g. ARINC/APEX rtos API standard for avionics). These approaches provide the ability to clearly model and organize components into architectures.

However, formal analysis and reasoning is gaining ever increasing importance due to application of software in critical applications. Many such applications ranging from bio-medical devices to sophisticated weapons systems are designed from components. Given the complexity of today's concurrent, distributed and networked software, it is extremely important to provide formal techniques and CASE tools for analysis and reasoning on local component properties as well as on global system properties.

Topics

We encourage submissions on formal techniques that aid reasoning, analysis and certification of embedded software and component-based software architectures.

In this context the following topics are of particular concern:

- component interoperability,
- contractually used components,
- interface compliancy (interface-to-interface and interface-to implementation),
- compliancy with synchronization constraints,
- temporal properties including liveness and fairness,
- safety, dependability, trustworthiness,
- quality properties (reliability, performance, timeliness),
- formal design methodologies and modeling techniques,
- formal methods for embedded real-time software,
- formal methods and pervasive computing.

Tools and techniques might involve (but are not limited to):

- logic-based approaches using interactive or automated theorem proving (e.g., B, Z, PVS, Coq),
- concurrency models (e.g., process calculi, refinement calculi, state machines, Petri-nets),
- type theory based reasoning of correctness, component composition frameworks,
- quantitative models on software.

Two types of submissions are considered: Long papers describing original research contributions and short papers giving experience reports on projects in the above context.

Submissions concentrating solely on specification techniques should involve an evaluation of the practical merit of their research and clearly state the analysis and reasoning techniques they enable. We also encourage work of a formal nature with immediate value to the industrial context.

Guest Speaker

Prof. Dr. Jose Luiz Fiadeiro, University of Leicester, England

Author instructions

Please use the latex style sheet information at http://math.tulane.edu/˜entcs/

Publication

Final versions of accepted full papers will be published by Elsevier in Electronic Notes in Computer Science (ENTCS). Accepted long (more than 10 pages) submissions are guaranteed publication in ENTCS. Accepted short (4 pages) submissions can be lengthened for another review round for inclusion in the ENTCS proceedings. The acceptance of a paper implies the active participation of at least on of the authors during the workshop.

Dates

Submission deadline:	14th December 2004
Notification of acceptance/rejection:	10th January 2005
Final version:	11th February 2005

Organizing Committee

- Juliana Kster Filipe, jkfilipe@inf.ed.ac.uk, School of Informatics, University of Edinburgh, UK (Contact, Co-chair)
- Iman H. Poernomo, iman@dcs.kcl.ac.uk, Department of Computer Science, King's College London, UK
- Ralf H. Reussner, reussner@informatik.uni-oldenburg.de, Department of Computing Science University of Oldenburg, Germany (Co-chair)
- Sandeep K. Shukla, shukla@vt.edu, Electrical and Computer Engineering, Virginia Tech, USA

Programme Committee

Angelo Corsaro	(University of California at Irvine, USA)
Frederic Doucet	(University of California at Irvine, USA)
Sabine Glesner	(University of Karlsruhe, Germany)
Guenter Kniesel	(University of Bonn, Germany)
Juliana Kster Filipe	(University of Edinburgh, UK)
Chris Ling	(Monash University, Australia)
Julia Padberg	(TU Berlin, Germany)
Iman H. Poernomo	(King's College London, UK)
Ralf H. Reussner	(University of Oldenburg, Germany)
Sandeep K. Shukla	(Virginia Tech, USA)
Jean-Pierre Talpin	(IRISA/INRIA, France)
Heike Wehrheim	(University of Oldenburg, Germany)

FESCA 2005 is supported by EASST, the European Association of Software Science and Technology (http://www.easst.org)

ACM support pending.

The homepage of the previous FESCA workshop in 2004 http://www.csse.monash.edu.au/fesca/index04.htm

European Association of Software Science and Technology EASST

Who are we?

EASST is a European non-profit Association that aims at promoting research, development and applications in the area of systematic and rigorous engineering of software and systems.

What are our aims?

Software and Systems Engineering does not receive the public recognition it deserves as one of the most advanced technologies with a great impact on Europe‘s economic and societel prosperity. This is due to a large extent to the low degree of visibility of the community. Especially research is scattered around a rather large number of communities, meetings in different conferences and workshops.

How do you benefit?

When joining us you enter a larger community and you will help to strengthen a new association that is aiming at a better visibility and recognition of your work.
When joining us you will benefit from a cross-fertilisation between a number of subcommittees in joint initiatives, meetings and activities.
When joining us you will have easy access to consolidated information collected from scattered sources.

How to participate?

All information will be made easily accessible by a number of electronic services.

Membership is for free.

Visit our Web-Site: http://www.isst.fhg.de

Statute of EASST

Name

European Association of Software Science and Technology

Location

The Association is located in Berlin/Germany.

Legal Status

The Association is a non-profit organization under German law (»gemeinnütziger eingetragener Verein«).

Purpose and Nature of Activities

The purpose of EASST is to promote the development of science and engineering on software intensive-systems, that play an increasing role in Europe´s way into the information society. It therefore supports education and qualification in software science and engineering, advices decision makers on appropriate measures, and informs the general public on the impact of technology developments.

The Association will

1. organize the exchange of information and spread research results by appropriate means to the community
2. provide help in the coordination of initiatives and projects in the area
3. organize and/or sponsor conferences like ETAPS and other professional meetings
4. coordinate its activities with other professional associations with the goal to give birth to a joint European association in informatics.

Membership

Ordinary membership in the association is open to individuals and legal entities, including other professional associations that support the goals of the EASST.
Associated membership may be obtained by members of other professional societies after proper agreement between them and EASST. Membership applications are requested in written form as determined by the board.

Membership Fee

A membership fee is not collected initially and may be collected later on, only after a decision taken by the membership at large.

Termination of Membership

Membership may be terminated by the member's resignation.
Membership will be terminated if the interest of the member in the membership in EASST vanishes. Indication of lost interest is abstention from decisions taken in EASST in electronic ballots for more than four times consecutively.
Membership will also be terminated if a membership fee due according to decision taken by the membership at large is not paid after its invoicing and after a second request.

Organs and Officers

General Assembly
The membership at large constitutes the general assembly of EASST. The General Assembly elects members of the board of EASST once every two years.
The General Assembly meets at least once a year to receive the annual report of the board including a financial and an activities report. An acceptance vote is expected four weeks after the issue of the report.
The General Assembly votes on the statutes of EASST not later than one year after its constitution, and on further amendments to the statute as well as on the dissolution of the association.

Board

The Board consists of the president, the vice president, the treasurer, the secretary, and four other board members without a particular portfolio.

Voting

Voting takes place in written form as determined by the board. The acceptance/rejection of the statutes, the amendment of the statute and the dissolution of the association require a two thirds majority of the members taking part in the vote.

Termination

In the event of the dissolution of the Association any remaining fund shall be disposed of in a manner determined by the General Assembly so as to support the purposes of EASST.

Application Form

I wish to become a member of EASST.

Please complete the following:

Name, First Name_____

Title_____

Company/University_____

Position_____

Street_____

Postal Code, City_____

Phone_____

Fax_____

E-Mail_____

Date and Signature_____

and return this form as soon as possible to:

EASST c/o
Herbert Weber
Fraunhofer-Institut für Software- und Systemtechnik
Mollstraße 1
D-10178 Berlin

Fax: +49 (0) 30/2 43 06-1 99
E-Mail: herbert.weber@isst.fhg.de