

THE EASST NEWSLETTER

Volume 14

December 2006



European Association of Software Science and Technology



EASST Board:

Prof. Dr. Tiziana Margaria (President),

email : margaria@cs.uni-potsdam.de

Prof. Dr. Hartmut Ehrig (Vice-President),

email : ehrig@cs.tu-berlin.de

Dr. Michael G. Hinchey (Treasurer),

email : Michael.G.Hinchey@nasa.gov

Dr. Julia Padberg,

email : padberg@cs.tu-berlin.de

Prof. Dr. Marie-Claude Gaudel,

email : Marie-Claude.Gaudel@lri.fr

Dr. Maura Cerioli,

email : cerioli@disi.unige.it

Dr. Michel Wermelinger (column editor),

email : m.a.wermelinger@open.ac.uk

Prof. Susanne Graf,

email : Susanne.Graf@imag.fr

Homepage of EASST:

<http://www.easst.org>

Subscription:

EASST NEWSLETTER is distributed among the members of EASST, the *European Association of Software Science and Technology*. If you are not yet a member of EASST, but you wish to receive the EASST NEWSLETTER, then you are kindly invited to become a member! Note, there are **no** membership fees. The application form can be found on the last page.

Or apply online at <http://www.easst.org>

Editor:

Dr. Julia Padberg

Technische Universität Berlin

Fakultät IV - Elektrotechnik und Informatik

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 | Julia Padberg | 4 |
| To Store or Not To Store. Reloaded: Reclaiming Memory on Demand | Moritz Hammer, Michael Weber | 5 |
| Review on the textbooks Software Engineering 1-3 by Dines Bjørner | Hartmut Ehrig | 12 |
| Report on MoDELS/UML 2006 and on GIIS meeting 2006 | Maura Cerioli | 15 |
| FMICS 2006 - 11th International Workshop on Formal Methods for Industrial Critical Systems | Luboš Brim, Martin Leucker | 18 |
| Report on Establishing The Electronic Communications of the EASST | Julia Padberg | 19 |
| The EASST Flyer | | 22 |
| Application Form | | 25 |

WELCOME

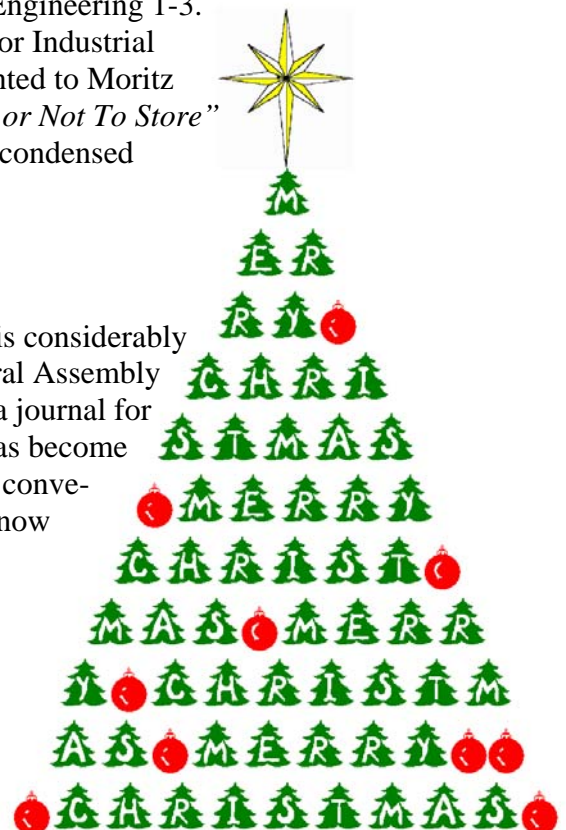
Dear Members of EASST,

this is the Xmas issue of the EASST newsletter and again we have exciting contributions. There is a review on Dines Bjørner's textbooks on Software Engineering 1-3. During the 12th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2007) the EASST award was presented to Moritz Hammer and Michael Weber for their contribution "To Store or Not To Store" *Reloaded: Reclaiming Memory on Demand*. Here you find a condensed version of this paper. Maura Cerioli has prepared a report on MoDELS/UML 2006 and on the GIIS meeting 2006.

Maybe you have noticed when downloading this issue that it is considerably less in size than the previous issues. In March 2006 the General Assembly has decided that in addition to the newsletter there should be a journal for EASST as well. One of the reasons was that the newsletter has become too crowded, so that sending it via email has not always been convenient. So I am happy to announce that the EASST-journal is now available: The Electronic Communications of the EASST (ECEASST <http://www.easst.org/eceasst>). A short report on that journal can be found in this issue as well.

I wish you a Merry Christmas
and a Happy New Year

Julia Padberg





“To Store or Not To Store” Reloaded: Reclaiming Memory on Demand¹

Moritz Hammer* and Michael Weber**²

*Ludwig-Maximilians-Universität München, Institut für Informatik
hammer@pst.ifi.lmu.de

**CWI, Dept. of Software Engineering, Amsterdam, The Netherlands
Michael.Weber@cwi.nl

1 Introduction

Model checking of industrial-scale models is usually restricted by memory size. The research addressing this issue can be divided into two groups: Automatically making models smaller by abstraction, and getting more out of the memory available. While the former direction has seen significant improvements with the advent of CEGAR model checking [3], large-scale model checking still remains an important area of research.

In explicit-state model checking, memory size is mostly restricted by the hash table used to detect state revisits. Among the many solutions to this problem, disk-based model checking and lossy hash tables have been proposed. Disk-based model checking basically tries to get a bigger hash table by swapping states to disk. As disk space is much cheaper than RAM, larger state spaces can be checked on normal systems. Disk storage is of course much slower, especially in random-access mode, which establishes the need for adapted algorithms.

Lossy hashing makes use of the observation that the hash table is used for avoiding revisits of parts of the state space that have been visited before. While revisiting is costly with respect to runtime, it does not void the correctness of the search result, as long as the hash table is sufficient to ensure termination. Therefore, it can be attempted to remove states from the lossy hash table in order to reclaim their memory and make room for further, hopefully newly discovered states. Behrmann et al. [1] claim that only 10% of the state space needs to be stored, allowing for much larger search spaces; however this will lead to extensive revisits in our experience.

Both directions have been given much research, but they have not yet been combined. Disk-based model checking usually does not make attempts to check a model in RAM only, should it turn out to

¹A full version of this paper is due to appear in the proceedings of FMICS 2006

²This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.000.05N09



be small enough. Lossy hashing, on the other hand, does not attempt to avoid revisits. By combining both strategies, we obtain an algorithm that avoids extensive revisits by doing disk lookups, but runs in RAM only as long as this is sufficient. This leads to a smoothly degrading algorithm, which adaptively changes its use of memory: For smaller models, it is little different from a RAM-only model checker, and it gradually becomes a disk-based model checker if memory becomes scarce.

Contributions

We develop an external-memory algorithm for explicit-state space generation in which we make novel use of well-known data structures. In the following sections, we present three main ingredients to make our algorithm practical in a disk-based setting. For reduction of expensive disk input and output (I/O) queries we use a combination of lossy hash tables and Bloom filters [2] which turn out to nicely complement each other. The third ingredient comes into play for disk operation that cannot be avoided: grouping them into bulk operations which can be performed linearly (instead of randomly) on disk not only improves performance, it also enables the use of compression techniques.

Our experiments on industrial-size case studies show that we can easily handle state spaces of more than 10^9 states. A distinguishing feature of our algorithm is that it performs well if entire state vectors of all states are to be stored on disk, i.e. we do not apply lossy compression. The stored states can be used in off-line analysis, and graph minimization with existing tools [5, 6].

2 Safe Lossy Hashing

In an explicit model checker, the hash table containing the *closed set*, i. e. the set of states already visited, is the most memory-demanding component. In fact, in the SPIN model checker [7], it is the only component to allocate memory once the search has been started. As such, it becomes the limiting factor for model size, and various ways of extending it have been proposed: using under- and over-approximations of the visited set, or making it larger by swapping states to disk.

In our approach, we maintain a chained hash table in memory. But once the table fills up, we select *victim states* and swap them to disk, thus freeing the memory they (and the hash table buckets that contain them) claim. The hash table then still serves as an under-approximation of the closed set, i.e. any state that is found to be stored in the hash table is known to have been visited previously, but a miss must be checked against disk.

The selection of victim states is done based on heuristics, for efficiency reasons. We compared a number of such strategies, which can be divided into two categories, depending on whether they require preceding (model-specific) analysis or can be done solely while performing state space exploration. Behrmann et al. report on model-specific analyses [1], while we aim to stay model-independent for the benefit of reuse, thus concentrating on the latter group.

As heuristics, we considered purely random choice, and choice based on incoming and outgoing edges, state *age*, and combinations. However, in our experiments (for example, Table 1) it turns out that for the case we are interested in, namely when the lossy hash table size is much smaller than the actual size of the full state space, the random replacement strategy performs not much worse than any of the more sophisticated approaches. Its lack of extra book keeping makes it very competitive.



| LHT size | Cache failures for Reclaiming Strategy | | | | |
|--|--|-----------|-----------|-----------|-------------|
| | Random | Incoming | Outgoing | Age | Combination |
| <i>Dining Philosophers</i> ($n = 9$): 4,685,071 states, 12,234,622 transitions | | | | | |
| 2% | 7,569,757 | 7,799,983 | 7,965,160 | 7,657,372 | 7,702,798 |
| 25% | 3,282,324 | 4,913,862 | 4,366,858 | 3,251,002 | 5,143,714 |
| 80% | 178,254 | 429,148 | 403,077 | 9,481 | 9,481 |
| <i>LUNAR scenario 4(b)</i> : 3,335,917 states, 3,923,209 transitions | | | | | |
| 2% | 225,960 | 233,010 | 259,499 | 228,485 | 194,332 |
| 25% | 70,126 | 62,733 | 97,652 | 59,272 | 43,221 |
| 80% | 8,124 | 4,861 | 13,249 | 64 | 4 |

Table 1: Comparison of *cache failures* with different reclaiming strategies. Column “LHT size” indicates the size of the lossy hash table with respect to the total state space size.

For storing states on a disk, a data structure supporting random access may seem the best choice. However, due to the large number of requests and the unpredictable distribution of states, this structures (like a hash table on disk) tend to become prohibitively slow due to disk latency times: As a lookup of any given state is likely to require the reloading of a new block on disk, each state lookup is paid for by the disk’s latency time. We thus take the approach taken by most disk-based model checking tools in some way or other: We group our read and write accesses, so that entire *pages of states* can be read and written at once. We can easily achieve this when writing states by collecting them in an in-memory staging area prior to actually writing them to disk. For reading states, we need to collect states that have been found to be possibly unvisited in a *candidate set*, which will eventually be checked against disk at once. This technique is known as *delayed duplicate detection* (DDD), as for a conflicting state, resolution is postponed until it may be conducted efficiently.

2.1 Single-Successor States

Apart from reclaiming memory, we can take another measure towards keeping memory requirements of the hash table tractable: We may choose only to store states that have more than one successor. While states with one outgoing transition may be target of multiple incoming transitions, the cost of revisiting them is limited: the previously visited chain of single-successor states is followed again, until a state with multiple successors is found and the revisit is detected. This optimization is used for example in the SPIN model checker [7]. It is very easy to implement and independent of the state space generator, as it does not need to take the actual semantics of the state vector into consideration.

We found that the single-successor optimization in some cases vastly reduces the number of states that need to be stored, yet it comes not always for free: in our experiments we sometimes witnessed a high number of revisits, resulting from chains of states with in-degree greater than one. However, note that even if states are revisited, they are not stored to disk again, nor do they produce additional disk lookups. If the state space generator is fast enough, the cost of calculating revisited states is easily outmatched by



the decreased cost in storing these states.

The single-successor optimization is very effective on models exhibiting a high degree of determinism, that is, in many states exactly one transition can be executed. Models created by experts are often in this class [9], however this is also dependent on the modeling problem at hand. Means to manually make models more deterministic are language constructs like atomic regions and mutexes.

It is worth noting that such mostly deterministic models often do not benefit much from *partial order reduction* techniques, an otherwise very effective technique for state space reduction.

3 Bloom-Filter Cache

Lossy hashing enables us to reclaim memory, in case the state space becomes too large to be kept in memory. Using disk lookups will prevent us from repeatedly searching parts of the state space. However, once states have been removed from the in-memory hash table, we can no longer predict whether a state that was not found in the hash table has indeed not been visited yet, or has been swapped to disk. A query to the lossy hashing will be answered with either “visited” or “don’t know”, and the “don’t know” answers need to be checked against disk.

Since this procedure entails a disk lookup for each unvisited state, it is quite costly. An inexpensive way of reducing the number of lookups for new states is offered by Bloom filters [2]. Basically, a Bloom filter is a set representation consisting of an array of m bits and k hash functions f_1, \dots, f_k . To add an element e , the bits $(f_1(e) \bmod m), \dots, (f_k(e) \bmod m)$ are set.

While coverage of Bloom filters is notably good and cheap [4], the filter maintains an over-approximation of the set of visited states, and its use in, e.g., SPIN [8] is unsafe. If we were to rely purely on the Bloom filter, hash collisions might prevent some parts of the state space to be visited. Dual to the lossy hashing, a *miss* in the Bloom filter will be sufficient to know that a state is unvisited yet, whereas a *hit* can be a *false positive* due to hash collisions. Thus, our Bloom filter will answer a state query with either “unvisited” or “don’t know”.

Together with a lossy hash table, we now get the best of both worlds: an *in-memory cache* that gives a three-valued answer whether a state was either “visited” (a hit in the lossy hash table), “unvisited” (a miss in the Bloom filter), or “don’t know” if both data structures fail to give a conclusive answer. The cost of this combined query filter is little more than for lossy hashing alone. We need, however, memory to organize the states which are subject to DDD.

Fig. 1 shows the complete algorithm in pseudo-code. While the order in which the Bloom filter and the lossy hashing are queried is arbitrary with respect to the correctness of the algorithm, it is more efficient to precede the lossy hashing check by the Bloom filter check, which is cheaper and may obviate a check of the lossy hashing (which sometimes involves the processing of a chain of hash buckets and comparing state vectors).

4 Improving disk I/O

While the caches help to minimize the amount of disk lookups required, disk I/O remains to be very expensive, especially for large models where the caches eventually degrade. Linear reads help to reduce



```

open ← {initialState}
candidate ← ∅
disk ← ∅
while (true) do
  if open = ∅ then diskLookup() fi
  State s ← open.removeState();
  for s' ∈ succ(s) do
    if (bloom.isUnvisited(s'))
      then addToOpen(s')
      else if ¬lossy.isVisited(s')
        then candidate ←
          candidate ∪ {s'}
    fi
  fi
od
od

proc addToOpen(State s) ≡
  open ← open ∪ {s}

  bloom.add(s)
  if lossy.isFull()
    then State s' ← lossy.reclaim()
    disk ← disk ∪ {s'}
  fi
  lossy.add(s)
  .
proc diskLookup() ≡
  for s ∈ disk do
    if s ∈ candidate
      then candidate ← candidate \ {s}
    fi
  od
  if candidate = ∅ then terminateSearch() fi
  for s ∈ candidate do
    addToOpen(s)
  od
  .

```

Figure 1: Pseudo-code for the RECLAIM algorithm

the influence of disk latency, but the I/O throughput remains a bottle-neck. In order to minimize this throughput, the disk-based $MUR\varphi$ implementations employ hash compaction, i.e. not store the state itself, but a hash signature instead, which can be seen as a lossfull compression. While the compression ratio is very good (usually, 40 or 48 bits are used, while uncompressed states usually require 500 bytes and more), hash signature collisions can lead to unvisited states, and the state information is lost for those states written to disk and removed from memory.

We can, however, take advantage of the bulk character of our disk I/O operations: We use paging to write sets of states to disk at once, and also read sets of states during disk lookups. The size of such sets can be chosen arbitrarily, and they are a good target for dictionary-based compression, as used by the Lempel-Ziv LZ77 algorithm. We use the ZLIB library and usually achieve a 20-fold compression, in extreme cases even 100-fold. Dictionary-based compression performs well as most states do not differ much and should contain many repetitive sections.

5 Implementation and Results

In our prototype implementation of our approach, we maintain different sets of states:

- the *open set* consists of all states that have been built, found to be unvisited but not yet processed,
- the *candidate set* is a set of states that need to be checked against the disk to verify that they have not yet been visited,



| Model | States | | Edges | Time [h] | ZLIB compr. | Uncompr. stored | Bloom failures | Cache |
|----------------|---------|--------|--------|----------|-------------|-----------------|----------------|--------|
| | visited | stored | | | | | | |
| GIOPI | 192.9M | 162.5M | 664.6M | 13:34:21 | 4.81% | 79.2GB | 363.3K | 91.2M |
| HUGO: Hot fail | 555.6M | 205.3M | 864.9M | 15:18:16 | 3.79% | 166.9GB | 127.9K | 32.1M |
| LUNAR 4(d) | 1.3G | 248.3M | 1.9G | 35:37:29 | 5.27% | 153.0GB | 1.7M | 150.8M |
| LUNAR 4(f) | 1.6G | 334.6M | 2.6G | 38:36:02 | 5.73% | 230.0GB | 12.7M | 387.0M |

Table 2: Runs for large models. States visited are all states, including single-successor states, whereas states stored are only states with more than one successor.

- the *reclaim set* is a set of states that are stored in the lossy hashing, but have been scheduled for writeout to disk in order to save memory.

New states are built as either the initial state or successor states to a state taken from the open set. These states are passed to the main search algorithm which then queries the Bloom filter and the lossy hashing to obtain a three-valued answer: either the state is not visited, in which case it is added to the open set, or it has been visited already, in which case it is discarded, or we cannot obtain a definitive answer from the caches, in which case we add the state to the candidate set.

States that are added to the open set are added to the caches as well. By adding new states to the caches, the lossy hashing eventually runs full, and a reclaim set is chosen. This set is written to disk and states from the set are removed if room is needed for new states. Once the open set becomes empty, or the candidate set becomes too large, a disk lookup is triggered. We use the seemingly inferior technique of checking the complete set of states written to disk in a linear fashion, comparing each state against an index built from the candidate set. However, this bulk reading decreases the impact of disk latency time on a single lookup. This technique, known as “delayed duplicate detection”, is employed by most disk-based model checkers. Unvisited states are added to the open set, while visited states are discarded.

5.1 Results

Table 2 shows the results of some large models. For all four models, the single-successor criterion works quite well, as can be seen by comparing the number of states actually stored to the number of states visited, which is the sum of the former number and those states discarded due to the single-successor criterion. The visited states number is an over-approximation of the actual number of states in the state space, as is the number of edges, but we still need to store only a fraction of states. Also, ZLIB compression performs very well. We consider both properties to be specific for industrial protocols, which are not too nondeterministic (comparing the number of edges to the number of states covered) and rely on rather large state vectors (more than 500 bytes for all models, with 895 bytes for the hot fail-over protocol).



6 Conclusions

In this paper, we presented a novel algorithm for state space exploration using external memory. It makes novel use of a combination of Bloom filters, lossy hash tables and compression to reduce disk I/O, while still ensuring that all reachable states are indeed explored. Additionally, states are stored to disk for off-line post-processing. Our experiments confirm that our method is practical. We can easily handle large case studies in the order of 10^9 states.

As future work, we would like to address how to combine these algorithms with checking of arbitrary temporal formulas. We are confident that we can build on work done for parallel and distributed model checking algorithms there. This also leads to the next logical step, namely combining disk-based and distributed model checking, allowing faster results, and possibly pushing towards the Tera-state (10^{12}) scale.

References

- [1] Gerd Behrmann, Kim Guldstrand Larsen, and Radek Pelánek. To store or not to store. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 433–445. Springer, 2003.
- [2] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
- [3] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and A. Prasad Sistla, editors, *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.
- [4] Peter C. Dillinger and Panagiotis Manolios. Fast and accurate bitstate verification for SPIN. In *SPIN*, volume 2989 of *LNCS*. Springer, 2004.
- [5] Hubert Garavel, Frederic Lang, and Radu Mateescu. An overview of CADP 2001. *EASST Newsletter*, 4:13–24, August 2002.
- [6] Fredrik Holmén, Martin Leucker, and Marcus Lindström. UppDMC – a distributed model checker for fragments of the μ -calculus. In Lubos Brim and Martin Leucker, editors, *Proc. 3rd PDMC*, volume 128(3) of *ENTCS*. Elsevier Science Publishers, 2004.
- [7] Gerald J. Holzmann. *The SPIN model checker: primer and reference manual*. Addison-Wesley, Boston, MA 02116, September 2003.
- [8] Gerard J. Holzmann. An analysis of bitstate hashing. *Form. Methods Syst. Des.*, 13(3):289–307, 1998.
- [9] R. Pelánek. Evaluation of on-the-fly state space reductions. In *Proc. of Mathematical and Engineering Methods in Computer Science (MEMICS'05)*, pages 121–127, 2005.



Review on the textbooks Software Engineering 1-3 by Dines Bjørner

Hartmut Ehrig

Technical University of Berlin
Institute for Software Engineering and Theoretical Computer Science
Franklinstr. 28/29, D-10587 Berlin
ehrig@cs.tu-berlin.de

Dines Bjørner is certainly one of the most well-known protagonists of formal methods in software and system engineering. Already in the 70s he was one of the fathers of the Vienna Definition Method and Language advocating formal specifications for software development. More recently he has also been one initiator of the RAISE Method and the RAISE Specification Language RSL. In fact, RSL plays an important role in the 3 volumes Software Engineering 1-3 published as texts in Theoretical Computer Science by Springer in 2006.

Although there is already a large variety of text books on software engineering available, it is certainly a pleasure to read and study the 3 volumes by Dines Bjørner. All the relevant concepts from software engineering are systematically introduced and the role of formal methods is carefully explained. The reviewer shares the point of view of the author that formal techniques apply in all phases, stages and steps of software engineering, and in the development of all kinds of software. The main question, however, is how to present formal methods to people in software engineering, especially to those without solid background in mathematics. The answer given by the author is “Formal Techniques Light”. This means basically 3 steps: “1. Start by being systematic. 2. Specify crucial facts formally. 3. Program Code from there.” The first 2 steps are presented carefully for a large variety of concepts and examples covered in the 3 volumes. For step 3, however, there is essentially only a hint that tools can be provided to translate model oriented specifications in RSL into constructs of programming languages like C, C++, C#, Java, and Standard ML.

Let us have a closer look at each of the volumes. In volume 1 with subtitle “Abstraction and Modelling” we have in part I a careful introduction into the aims and objectives of the 3 volumes, in part II an informal, but systematic treatment of several areas of discrete mathematics, and in parts III and IV an introduction into RSL. Finally in part IV the concepts of applicative, imperative, and concurrent specification programming are carefully explained and shown how they are supported by RSL. RSL is justified as suitable specification language for this purpose, because it is close to discrete mathematics, allows expressing the imperative



specification style, can handle expressions of concurrency, has a strong flavour of algebraic specification languages and allows to structure its specifications in a modular fashion. Remarkable in part II is the kind of informal, but systematic treatment of sets, functions, algebra and logic, which almost avoids any mathematical definition. It is coherent with the standard definitions, except of the notion of injective function, which – surprisingly – is not required to be “one-to-one”, but “non-surjective”.

Volume 2 with subtitle “Specification of Systems and Languages” starts in part I with an RSL primer summarizing all RSL-concepts introduced in volume 1, in part II and III specification facets like hierarchies and composition, denotations and computations, configuration, content and states as well as time and space are carefully introduced. Part IV concerns the linguistic concepts of syntax, semantics and pragmatics, which can be summarized as semiotics. Further specification techniques like modularization, automata and machines are presented in part VI and other specification techniques like Petri nets, message and live sequence charts, statecharts and quantitative models of time in part V. Finally interpreter and compiler definitions are discussed carefully in part VII including different kinds of simple applicative imperative, modular and parallel languages.

The chapters on visual specification techniques in part VI are authored by Christian Krog Madsen, a former student of Dines Bjørner. Core concepts of these techniques are introduced on an informal level and then syntax, static and dynamic semantics are given in RSL. This is called “UML”-ising formal techniques.

The main question “What is Software Engineering?” is addressed in volume 3 with subtitle “Domains, Requirements, and Software Design”. The answer to this question is summarized by Dines Bjørner in his “Triptych of Software Engineering”. The main idea of this triptych is that software development is an iterative process involving domain engineering, requirements engineering and software design, where all phases should be based on formal techniques. Bjørner claims – and the reviewer agrees in principle – that “developing software without formal techniques is like sailing the high seas without knowing how to compute the current longitude”. According to this triptych paradigm carefully discussed in part I, domain engineering, requirements engineering and computing systems design (including software design) is presented on a conceptual level with a large variety of examples and case studies specified in RSL in parts IV, V, and VI of volume 3 respectively. In the closing part VII it is a pleasure to read how Dines Bjørner is fighting against several myths about formal techniques of software engineering.

In summary, the 3 volumes with altogether about 2.250 pages – present a most interesting view of software engineering, which is certainly different from most other textbooks, especially from those focusing on implementation, testing and management issues within



software engineering. A suitable subtitle for all 3 volumes could be “The role of formal methods in software development”. The choice of RSL as formal specification technique is consistent with the aims of the author, although other protagonists of formal methods in software engineering might prefer other specification or modelling techniques. Moreover, the strong claim of model-oriented software development is widely supported in software engineering these days, where in most cases; however, object-oriented software development based on modelling and metamodelling techniques in the sense of UML and other visual techniques is predominant.

From the formal methods point of view the reviewer agrees with Bjørner that the lack of formal semantics for main parts of UML is still unsatisfactory, but “UML”-ising formal techniques is only one alternative. The reviewer also agrees with the author that the major shortcoming of his 3 volumes is the “all too brief coverage of correctness issues”, where the reader is referred to other literature. Although we regret that the reader is not guided how the show correctness of stepwise refinement from requirements to design, the principle “Formal Techniques Light” mentioned above is a convincing paradigm for the approach in these 3 volumes.

Altogether the 3 volumes can be highly recommended for software engineers in practice and students in software engineering courses in order to learn the basic principles of software development and how they can be supported by formal methods in a systematic way.

Report on MoDELS/UML 2006 and on GIIS meeting 2006

Maura Cerioli*

*DISI- Dip. di Informatica e Scienze dell'Informazione
Via Dodecaneso 35 - 16146 Genova - Italy

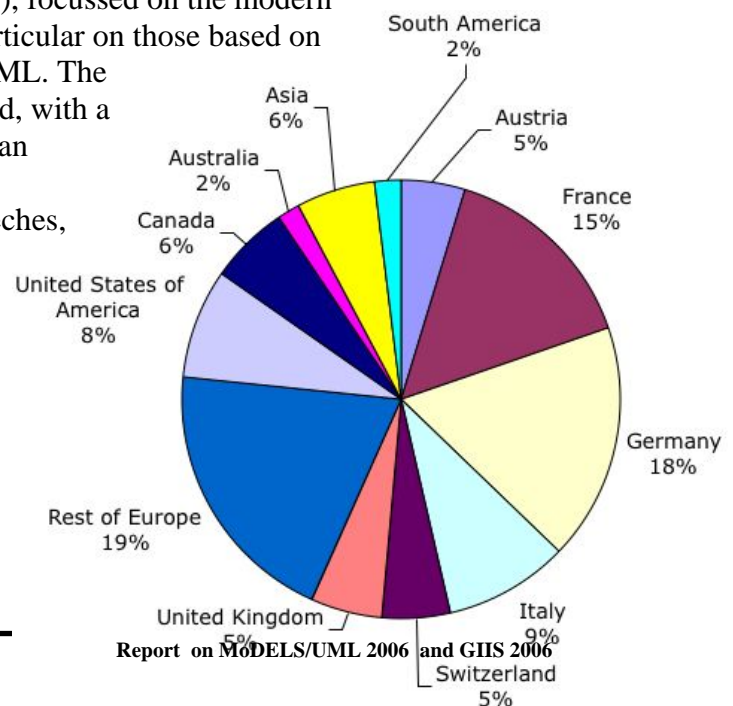
Abstract. Report on participation and scientific program of the conference MoDELS2006 yielded in Genova, Italy October 1-6 and on its satellite events, including the meeting of the Software Engineering Italian Group

Keywords: Modelling, Visual Notation, Software Engineering



The MoDELS/UML 2006 conference (<http://www.modelsconference.org/>) has taken place in Genova from 2 to 6 October, organized by DISI (Dipartimenti di Informatica e Scienze dell'Informazione). This international event, endorsed by ACM, IEEE, EASST, and by AICA (the Italian Association for Automatic Computing), focussed on the modern techniques of development of software, and in particular on those based on visual modelling notations like for instance the UML. The participants were over 300, from all over the world, with a larger presence from North American and European countries.

The scientific program included two keynote speeches, by Irun R. Cohen on Immune System Computation, presenting the model of the human body state used by the immune system in the same language used to model software systems and showing in this way the analogies between the two kinds of systems, and by Hassan Gomaa on A Software Modeling Odyssey: Designing Evolutionary Architecture-





centric Real-Time Systems and Product Lines, giving a fresco of the evolution of the main concepts of modelling. Moreover, two panels on different aspects of model-driven approaches and around 50 scientific presentations completed the program.

Before the main conference, there were 11 workshops, 3 symposia (whose participation was restricted to invited people) and 5 tutorials.

Workshops

- W1 Aspect-Oriented Modeling (O. Aldawud, W. Cazzola, T. Elrad, J. Gray, J. Kienzle, D. Stein)
- W2 CSDUML'06 - Critical Systems Development Using Modeling Languages (S.H. Houmb, G. Georg, J. Jürjens, R. France)
- W3 ATEM'06 - Metamodels, Schemas, Grammars and Ontologies for Reverse Engineering (J.M. Favre, D. Gašević, R. Lämmel, A. Winter)
- W4 Quality in Modeling (L. Kuzniarz, J.L. Sourrouille, R. Van Der Straeten, M. Staron, M. Chaudron, A. Förster, G. Reggio)
- W5 Model Driven Development of Advanced User Interfaces (A. Bödcher, H. Hußmann, A. Pleuß, S. Sauer, J Van den Bergh)
- W6 MARTES'06 - Modeling and Analysis of Real-Time and Embedded Systems (S. Gérard, S. Graf, J. Ober, Ø. Haugen, B. Selic)
- W7 OCL for (Meta-)Models in Multiple Application Domains (D. Chiorean, B. Demuth, M. Gogolla, J. Warmer)
- W8 MoDeV2a'06 - Perspectives on integrating MDA and V&V (D. Hearnden, J.G. Süß, N. Rapin, B. Baudry)
- W9 Model Size Metrics (B. Berenbach, A. Winsor Brown, B.H.C. Cheng, R. France, A. Neczwid, F. Weil)
- W10 Models@run.time (N. Bencomo, G. Blair, R. France)
- W11 MPM'06 - Multi-Paradigm Modeling: Concepts and Tools (T. Levendovszky, H. Giese)

Symposia

- Doctoral Symposium (R.G. Pettit, G. Arévalo)
- Educators Symposium (L. Kuzniarz)
- A Formal Semantics for UML Symposium (M. Broy, J. Dingel, A. Hartman, B. Rumpe, B. Selic)

Tutorials

- T1 Model-Driven Engineering of Distributed Systems (D. C. Schmidt, M. Völter)
- T2 Defining Domain-Specific Modelling Languages (J-P Tolvanen)
- T3 Model-Based Testing (A. Pretschner)



T4 Designing Software Product Lines with UML 2.0 (H. Gomma)

T5 Model Driven Engineering Basics using Eclipse (B. Trask, A. Roman)

The local industrial attendance to the tutorials has been encouraged by a series of seminars organized by DISI, DIST and the organization of ICT companies in Genova (Sezione Informatica di Confindustria Genova) introducing the new modelling techniques to practitioners.

At the same time as Models, also the third meeting of the GIIS (the Italian Group of Software Engineers) has been yielded in Genova, with an attendance of about 30 academic from all over Italy, and over 20 very short presentations giving a fresco of the topics currently researched in Italian universities, an invited talk and a panel with a large industrial presence. Half a day of the GIIS meeting was shared by the third edition of the Premio Perotto, which is an initiative by the Sezione Informatica di Confindustria Genova rewarding the most innovative ICT products with an award.



FMICS 2006 - 11th International Workshop on Formal Methods for Industrial Critical Systems

Luboš Brim *and Martin Leucker **

*Masaryk University, Brno, Czech Republic

**Technical University München, Germany

The 11th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 06) was held in Bonn, Germany, on August 26-27, 2006 as a satellite event to the 17th International Conference on Concurrency Theory (CONCUR 2006). The workshop, being also an annual meeting of the ERCIM Working Group, continued successfully the aim of FMICS workshop series - to promote the use of formal methods for industrial applications, by supporting research in this area and its application in industry. The emphasis in these workshops is on the exchange of ideas between researchers and practitioners, in both industry and academia.

Previous workshops were held in Oxford (March 1996), Cesena (July 1997), Amsterdam (May 1998), Trento (July 1999), Berlin (April 2000), Paris (July 2000), Malaga (July 2002), Trondheim (July 2003), Linz (September 2004), and Lisbon (July 2005). The 2006 workshop was organized by the Masaryk University Brno and the Technical University München. Forty two registered participants from academia and industry from about eleven countries attended the workshop.

This year the program committee received a record number of submissions. The 16 accepted regular contributions and 2 accepted tool papers, selected out of a total of 47 submissions, cover formal methodologies for handling large state spaces, model based testing, formal description and analysis techniques as well as a range of applications and case studies.

The workshop program included two excellent invited talks, respectively by Anna Slobodova from Intel on "Challenges for Formal Verification in Industrial Setting" and by Edward A. Lee from University of California at Berkeley on "Making Concurrency Mainstream".

The award for the best paper was granted this year to Michael Weber and Moritz Hammer for their excellent paper ""To Store or Not To Store" Reloaded: Reclaiming Memory on Demand" on the techniques for handling extremely large state spaces in model checking of computer systems. The award was granted with the support of the European Association of Software Science and Technology (EASST).

The final proceedings of the workshop will be published jointly with the 5th International workshop on Parallel and Distributed Methods in Verification (PDMC 2006) as post-proceedings in the Springer's Lecture Notes in Computer Science. The organizers wish to thank CONCUR for hosting the FMICS 2006 workshop and ERCIM for its financial support. Additionally, the organizers would like to thank EASST (European Association of Software Science and Technology), Faculty of Informatics, Masaryk University Brno and the Technical University München for supporting this event.



Report on Establishing The Electronic Communications of the EASST

Julia Padberg*

*Technische Universität Berlin

Fakultät IV - Elektrotechnik und Informatik

The formation of EASST was an important step to join the research and development activities concerning systematic and rigorous engineering of software and systems. The EASST newsletter is one medium which reports periodically on these activities. Furthermore, EASST supports several conferences and workshops like ETAPS/FASE and TACAS, as well as ICGT which come along with their separate proceedings. A journal which mirrors the EASST activities and provides a forum for practitioners, educators and researchers for disseminating innovative research on fundamental approaches to software engineering is presented hereby.

1 Title and Scope

The title is:

Electronic Communications of the EASST
ECEASST

URL: <http://www.easst.org/eceasst>

The journal's scope should be *Software Science and Technology* as in EASST which includes the satellite events of ETAPS, the European Joint Conferences on Theory and Practice of Software, of ICGT, the International conference on Graph Transformations, and of further EASST-supported conferences. This includes any aspect related to the systematic and rigorous engineering of software and systems. Currently, this journal is a medium for publishing special issues containing contributions from conferences and workshops on EASST-related topics and shall be extended towards special issues and regular contributions.

2 Open and On-line Access

Electronic Communications of the EASST is a peer-reviewed, scientific, and open access journal. It is a fully refereed journal that provides a forum for practitioners, educators and researchers for disseminating innovative research in the area of software and systems technology. The journal



is also intended as a medium to rapidly publish conference and workshop proceedings, with original and peer reviewed contributions. *ECEASST is an open access on-line journal, that provides unlimited and free access to all its contributions.* In this way, ECEASST grants full-text access to all the papers and supports the provision of fast and broad feedback to published work.

3 Editors and Publisher

There are three "managing editors" being mainly responsible for the new journal. Later the managing editors should be appointed by the EASST Board for a period of 3-5 years.

For the moment these are:

- Tiziana Margaria, University of Postdam
- Julia Padberg, Technische Universität Berlin
- Gabriele Taentzer, Philipps-Universität Marburg

Further members of the Editorial Board are presently:

- Maura Cerioli, DISI - Dipartimento di Informatica e Scienze dell'Informazione
- Antonio Cerone, UNU-International Institute for Software Technology
- Hartmut Ehrig, Technical University Berlin
- Marie-Claude Gaudel, INRIA-Futurs
- Susanne Graf, VERIMAG
- Reiko Heckel, University of Leicester
- Michael G. Hinchey, NASA
- Heinrich Hussmann, Ludwig-Maximilians-Universitt Mnchen
- Jens Knoop, Technische Universitt Wien
- Fernando Orejas, Universitat Politcnica de Catalunya
- Andrzej Tarlecki, Warsaw University

The EASST board and the EASST representatives have been considered as an initial pool from which people were invited. Another pool could be the set of all FASE-PC chairs and FASE invited speakers from all ETAPS conferences up to ETAPS 2006.

The publisher of this journal is EASST as European Association.

4 Current and Future Volumes

There is the first volume now on-line available:

GraMoT 2006 The proceedings of *the Second International Workshop on Graph and Model Transformation* held in Brighton, United Kingdom, where it was a satellite event of the IEEE Symposium on Visual Languages and Human-Centric Computing, 2006.



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 societal 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, advises 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