Klaus Havelund
Manuel Núñez
Grigore Roşu
Burkhart Wolff (Eds.)

LNCS 4262

Formal Approaches to Software Testing and Runtime Verification

First Combined International Workshops
FATES 2006 and RV 2006
Seattle, WA, USA, August 2006
Revised Selected Papers

Springer

Klaus Havelund Manuel Núñez
Grigore Roşu Burkhart Wolff (Eds.)

Formal Approaches to Software Testing and Runtime Verification

First Combined International Workshops
FATES 2006 and RV 2006
Seattle, WA, USA, August 15-16, 2006
Revised Selected Papers

Springer

Volume Editors

Klaus Havelund
Jet Propulsion Laboratory
Laboratory for Reliable Software
4800 Oak Grove Drive, M/S 301-285, Pasadena, CA 91109, USA
E-mail: mn@sip.ucm.es

Manuel Núñez
Universidad Complutense de Madrid
C/ Prof. José García Santesmases, s/n.
Dep. Sistemas Informáticos y Computación Facultad de Informática
28040 Madrid, Spain
E-mail: mn@sip.ucm.es

Grigore Roşu
University of Illinois at Urbana-Champaign
Department of Computer Science
201 N. Goodwin, Urbana, IL 61801, USA
E-mail: grosu@cs.uiuc.edu

Burkhart Wolff
Information Security, ETH Zürich
ETH Zentrum, CH-8092 Zürich, Switzerland
E-mail: bwolff@inf.ethz.ch

Library of Congress Control Number: 2006937542

CR Subject Classification (1998): D.2, D.3, F.3, K.6

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-49699-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-49699-1 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11940197 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 4262

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Preface

Software validation is one of the most cost-intensive tasks in modern software production processes. The objective of FATES/RV 2006 was to bring scientists from both academia and industry together to discuss formal approaches to test and analyze programs and monitor and guide their executions. Formal approaches to test may cover techniques from areas like theorem proving, model checking, constraint resolution, static program analysis, abstract interpretation, Markov chains, and various others. Formal approaches to runtime verification use formal techniques to improve traditional ad-hoc monitoring techniques used in testing, debugging, performance monitoring, fault protection, etc.

The FATES/RV 2006 workshop selected 14 high-quality papers out of 31 submissions. Each paper underwent at least three anonymous reviews by either PC members or external reviewers selected by them. In addition to the 14 regular papers, the proceedings contain two papers corresponding to the invited talks by Wolfgang Grieskamp (Microsoft Research, USA) and Oege de Moor (Oxford University, UK).

This was the first time that the two workshops, FATES and RV, were held together. The success of this joint edition shows that the integration of these two communities can be profitable for both of them. Previous editions of these two events were held in the following places: FATES 2001 was held in Aalborg (Denmark) and FATES 2002 in Brno (Czech Republic). In both cases, the workshop was affiliated with CONCUR. FATES 2003 and FATES 2004 were held in Montreal (Canada) and Vienna (Austria), respectively, in affiliation with ASE. FATES 2005 was co-located with CAV in Edinburgh (UK). Since 2003, the FATES workshop proceedings have been published by Springer (LNCS series). In parallel, RV 2001 was held in Paris (France), followed by RV 2002 in Copenhagen (Denmark), and RV 2003 in Boulder (USA). These first three editions were affiliated with CAV. RV 2004 was held in Barcelona (Spain), affiliated with TACAS 2004. Finally RV 2005 was held in Edinburgh (UK), co-located with CAV. All previous editions of RV were published in Elsevier's *Electronic Notes in Theoretical Computer Science*. In addition, selected papers from RV 2001 and RV 2002 were published in Springer's journal *Formal Methods in System Design*, in issues 24(2) (March 2004) and 27(3) (November 2005), respectively.

We would like to express our gratitude to all the authors and invited speakers for their valuable contributions. We would also like to thank all members of the FATES/RV 2006 Program Committee and the additional reviewers for their efforts to accurately review the papers on time. Wolfgang Grieskamp supported the organization of the workshop by providing a PC projector and the printouts of these preliminary proceedings. In addition, Microsoft Research gave financial

support for the organization of the workshop. Finally, we would like to thank the local organization of FLoC 2006 for their help.

September 2006

Klaus Havelund
Manuel Núñez
Grigore Roşu
Burkhart Wolff

Lecture Notes in Computer Science

For information about Vols. 1–4237

please contact your bookseller or Springer

Vol. 4345: N. Maglaveras, I. Chouvarda, V. Koutkias, R.W. Brause (Eds.), Biological and Medical Data Analysis. XIII, 496 pages. 2006. (Sublibrary LNBI).

Vol. 4337: S. Arun-Kumar, N. Garg (Eds.), FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science. XIII, 430 pages. 2006.

Vol. 4333: U. Reimer, D. Karagiannis (Eds.), Practial Aspects of Knowledge Management. XII, 338 pages. 2006. (Sublibrary LNAI).

Vol. 4331: G. Min, B. Di Martino, L.T. Yang, M. Guo, G. Ruenger (Eds.), Frontiers of High Performance Computing and Networking – ISPA 2006 Workshops. XXXVII, 1141 pages. 2006.

Vol. 4329: R. Barua, T. Lange (Eds.), Progress in Cryptology - INDOCRYPT 2006. X, 454 pages. 2006.

Vol. 4326: S. Göbel, R. Malkewitz, I. Iurgel (Eds.), Technologies for Interactive Digital Storytelling and Entertainment. X, 384 pages. 2006.

Vol. 4325: J. Cao, I. Stojmenovic, X. Jia, S.K. Das (Eds.), Mobile Ad-hoc and Sensor Networks. XIX, 887 pages. 2006.

Vol. 4318: H. Lipmaa, M. Yung, D. Lin (Eds.), Information Security and Cryptology. XI, 305 pages. 2006.

Vol. 4313: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods. IX, 197 pages. 2006.

Vol. 4312: S. Sugimoto, J. Hunter, A. Rauber, A. Morishima (Eds.), Digital Libraries: Achievements, Challenges and Opportunities. XVIII, 571 pages. 2006.

Vol. 4311: K. Cho, P. Jacquet (Eds.), Technologies for Advanced Heterogeneous Networks II. XI, 253 pages. 2006.

Vol. 4307: P. Ning, S. Qing, N. Li (Eds.), Information and Communications Security. XIV, 558 pages. 2006.

Vol. 4306: Y. Avrithis, Y. Kompatsiaris, S. Staab, N.E. O'Connor (Eds.), Semantic Multimedia. XII, 241 pages. 2006.

Vol. 4305: A.A. Shvartsman (Ed.), Principles of Distributed Systems. XIII, 441 pages. 2006.

Vol. 4304: A. Sattar, B.-H. Kang (Eds.), AI 2006: Advances in Artificial Intelligence. XXVII, 1303 pages. 2006. (Sublibrary LNAI).

Vol. 4302: J. Domingo-Ferrer, L. Franconi (Eds.), Privacy in Statistical Databases. XI, 383 pages. 2006.

Vol. 4301: D. Pointcheval, Y. Mu, K. Chen (Eds.), Cryptology and Network Security. XIII, 381 pages. 2006.

Vol. 4300: Y.Q. Shi (Ed.), Transactions on Data Hiding and Multimedia Security I. IX, 139 pages. 2006.

Vol. 4296: M.S. Rhee, B. Lee (Eds.), Information Security and Cryptology – ICISC. XIII, 358 pages. 2006.

Vol. 4295: J.D. Carswell, T. Tezuka (Eds.), Web and Wireless Geographical Information Systems. XI, 269 pages. 2006.

Vol. 4294: A. Dan, W. Lamersdorf (Eds.), Service-Oriented Computing – ICSOC 2006. XIX, 653 pages. 2006.

Vol. 4293: A. Gelbukh, C.A. Reyes-Garcia (Eds.), MICAI 2006: Advances in Artificial Intelligence. XXVIII, 1232 pages. 2006. (Sublibrary LNAI).

Vol. 4292: G. Bebis, R. Boyle, B. Parvin, D. Koracin, P. Remagnino, A. Nefian, G. Meenakshisundaram, V. Pascucci, J. Zara, J. Molineros, H. Theisel, T. Malzbender (Eds.), Advances in Visual Computing, Part II. XXXII, 906 pages. 2006.

Vol. 4291: G. Bebis, R. Boyle, B. Parvin, D. Koracin, P. Remagnino, A. Nefian, G. Meenakshisundaram, V. Pascucci, J. Zara, J. Molineros, H. Theisel, T. Malzbender (Eds.), Advances in Visual Computing, Part I. XXXI, 916 pages. 2006.

Vol. 4290: M. van Steen, M. Henning (Eds.), Middleware 2006. XIII, 425 pages. 2006.

Vol. 4289: M. Ackermann, B. Berendt, M. Grobelnik, A. Hotho, D. Mladenič, G. Semeraro, M. Spiliopoulou, G. Stumme, V. Svatek, M. van Someren (Eds.), Semantics, Web and Mining. X, 197 pages. 2006. (Sublibrary LNAI).

Vol. 4288: T. Asano (Ed.), Algorithms and Computation. XX, 766 pages. 2006.

Vol. 4286: P. Spirakis, M. Mavronicolas, S. Kontogiannis (Eds.), Internet and Network Economics. XI, 401 pages. 2006.

Vol. 4285: Y. Matsumoto, R. Sproat, K.-F. Wong, M. Zhang (Eds.), Computer Processing of Oriental Languages. XVII, 544 pages. 2006. (Sublibrary LNAI).

Vol. 4284: X. Lai, K. Chen (Eds.), Advances in Cryptology – ASIACRYPT 2006. XIV, 468 pages. 2006.

Vol. 4283: Y.Q. Shi, B. Jeon (Eds.), Digital Watermarking. XII, 474 pages. 2006.

Vol. 4282: Z. Pan, A. Cheok, M. Haller, R.W.H. Lau, H. Saito, R. Liang (Eds.), Advances in Artificial Reality and Tele-Existence. XXIII, 1347 pages. 2006.

Vol. 4281: K. Barkaoui, A. Cavalcanti, A. Cerone (Eds.), Theoretical Aspects of Computing - ICTAC 2006. XV, 371 pages. 2006.

Vol. 4280: A.K. Datta, M. Gradinariu (Eds.), Stabilization, Safety, and Security of Distributed Systems. XVII, 590 pages. 2006.

Vol. 4279: N. Kobayashi (Ed.), Programming Languages and Systems. XI, 423 pages. 2006.

Vol. 4278: R. Meersman, Z. Tari, P. Herrero (Eds.), On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, Part II. XLV, 1004 pages. 2006.

Vol. 4277: R. Meersman, Z. Tari, P. Herrero (Eds.), On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, Part I. XLV, 1009 pages. 2006.

Vol. 4276: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, Part II. XXXII, 752 pages. 2006.

Vol. 4275: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, Part I. XXXI, 1115 pages. 2006.

Vol. 4274: Q. Huo, B. Ma, E.-S. Chng, H. Li (Eds.), Chinese Spoken Language Processing. XXIV, 805 pages. 2006. (Sublibrary LNAI).

Vol. 4273: I. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, L. Aroyo (Eds.), The Semantic Web - ISWC 2006. XXIV, 1001 pages. 2006.

Vol. 4272: P. Havinga, M. Lijding, N. Meratnia, M. Wegdam (Eds.), Smart Sensing and Context. XI, 267 pages. 2006.

Vol. 4271: F.V. Fomin (Ed.), Graph-Theoretic Concepts in Computer Science. XIII, 358 pages. 2006.

Vol. 4270: H. Zha, Z. Pan, H. Thwaites, A.C. Addison, M. Forte (Eds.), Interactive Technologies and Sociotechnical Systems. XVI, 547 pages. 2006.

Vol. 4269: R. State, S. van der Meer, D. O'Sullivan, T. Pfeifer (Eds.), Large Scale Management of Distributed Systems. XIII, 282 pages. 2006.

Vol. 4268: G. Parr, D. Malone, M. Ó Foghlú (Eds.), Autonomic Principles of IP Operations and Management. XIII, 237 pages. 2006.

Vol. 4267: A. Helmy, B. Jennings, L. Murphy, T. Pfeifer (Eds.), Autonomic Management of Mobile Multimedia Services. XIII, 257 pages. 2006.

Vol. 4266: H. Yoshiura, K. Sakurai, K. Rannenberg, Y. Murayama, S. Kawamura (Eds.), Advances in Information and Computer Security. XIII, 438 pages. 2006.

Vol. 4265: L. Todorovski, N. Lavrač, K.P. Jantke (Eds.), Discovery Science. XIV, 384 pages. 2006. (Sublibrary LNAI).

Vol. 4264: J.L. Balcázar, P.M. Long, F. Stephan (Eds.), Algorithmic Learning Theory. XIII, 393 pages. 2006. (Sublibrary LNAI).

Vol. 4263: A. Levi, E. Savaş, H. Yenigün, S. Balcısoy, Y. Saygın (Eds.), Computer and Information Sciences – ISCIS 2006. XXIII, 1084 pages. 2006.

Vol. 4262: K. Havelund, M. Núñez, G. Roşu, B. Wolff (Eds.), Formal Approaches to Software Testing and Runtime Verification. VIII, 255 pages. 2006.

Vol. 4261: Y. Zhuang, S. Yang, Y. Rui, Q. He (Eds.), Advances in Multimedia Information Processing - PCM 2006. XXII, 1040 pages. 2006.

Vol. 4260: Z. Liu, J. He (Eds.), Formal Methods and Software Engineering. XII, 778 pages. 2006.

Vol. 4259: S. Greco, Y. Hata, S. Hirano, M. Inuiguchi, S. Miyamoto, H.S. Nguyen, R. Słowiński (Eds.), Rough Sets and Current Trends in Computing. XXII, 951 pages. 2006. (Sublibrary LNAI).

Vol. 4257: I. Richardson, P. Runeson, R. Messnarz (Eds.), Software Process Improvement. XI, 219 pages. 2006.

Vol. 4256: L. Feng, G. Wang, C. Zeng, R. Huang (Eds.), Web Information Systems – WISE 2006 Workshops. XIV, 320 pages. 2006.

Vol. 4255: K. Aberer, Z. Peng, E.A. Rundensteiner, Y. Zhang, X. Li (Eds.), Web Information Systems – WISE 2006. XIV, 563 pages. 2006.

Vol. 4254: T. Grust, H. Höpfner, A. Illarramendi, S. Jablonski, M. Mesiti, S. Müller, P.-L. Patranjan, K.-U. Sattler, M. Spiliopoulou, J. Wijsen (Eds.), Current Trends in Database Technology – EDBT 2006. XXXI, 932 pages. 2006.

Vol. 4253: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part III. XXXII, 1301 pages. 2006. (Sublibrary LNAI).

Vol. 4252: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part II. XXXIII, 1335 pages. 2006. (Sublibrary LNAI).

Vol. 4251: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part I. LXVI, 1297 pages. 2006. (Sublibrary LNAI).

Vol. 4250: H.J. van den Herik, S.-C. Hsu, T.-s. Hsu, H.H.L.M. Donkers (Eds.), Advances in Computer Games. XIV, 273 pages. 2006.

Vol. 4249: L. Goubin, M. Matsui (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2006. XII, 462 pages. 2006.

Vol. 4248: S. Staab, V. Svátek (Eds.), Managing Knowledge in a World of Networks. XIV, 400 pages. 2006. (Sublibrary LNAI).

Vol. 4247: T.-D. Wang, X. Li, S.-H. Chen, X. Wang, H. Abbass, H. Iba, G. Chen, X. Yao (Eds.), Simulated Evolution and Learning. XXI, 940 pages. 2006.

Vol. 4246: M. Hermann, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning. XIII, 588 pages. 2006. (Sublibrary LNAI).

Vol. 4245: A. Kuba, L.G. Nyúl, K. Palágyi (Eds.), Discrete Geometry for Computer Imagery. XIII, 688 pages. 2006.

Vol. 4244: S. Spaccapietra (Ed.), Journal on Data Semantics VII. XI, 267 pages. 2006.

Vol. 4243: T. Yakhno, E.J. Neuhold (Eds.), Advances in Information Systems. XIII, 420 pages. 2006.

Vol. 4242: A. Rashid, M. Aksit (Eds.), Transactions on Aspect-Oriented Software Development II. IX, 289 pages. 2006.

Vol. 4241: R.R. Beichel, M. Sonka (Eds.), Computer Vision Approaches to Medical Image Analysis. XI, 262 pages. 2006.

Vol. 4239: H.Y. Youn, M. Kim, H. Morikawa (Eds.), Ubiquitous Computing Systems. XVI, 548 pages. 2006.

Vol. 4238: Y.-T. Kim, M. Takano (Eds.), Management of Convergence Networks and Services. XVIII, 605 pages. 2006.

Table of Contents

Invited Talks

Regular Papers

Multi-paradigmatic Model-Based Testing

Wolfgang Grieskamp

Microsoft Research, Redmond, WA, USA
wrwg@microsoft.com

Abstract. For half a decade model-based testing has been applied at Microsoft in the internal development process. Though a success story compared to other formal quality assurance approaches like verification, a break-through of the technology on a broader scale is not in sight. What are the obstacles? Some lessons can be learned from the past and will be discussed. An approach to MBT is described which is based on *multi-paradigmatic* modeling, which gives users the freedom to choose among programmatic and diagrammatic notations, as well as state-based and scenario-based (interaction-based) styles, reflecting the different concerns in the process. The diverse model styles can be combined by model composition in order to achieve an integrated and collaborative model-based testing process. The approach is realized in the successor of Microsoft Research's MBT tool Spec Explorer, and has a formal foundation in the framework of action machines.

1 Introduction

Testing is one of the most cost-intensive activities in the industrial software development process. Yet, not only is current testing practice laborious and expensive but often also unsystematic, lacking engineering methodology and discipline, and adequate tool support.

Model-based testing (MBT) is one of the most promising approaches to address these problems. At Microsoft, MBT technology has been applied in the production cycle since 1999 [1,2,3,4,5]. One key for the relative success of MBT at Microsoft is its attraction for a certain class of well-educated, ambitious test engineers, to which it is one way to raise testing to a systematic engineering discipline.

However, at the larger picture, an estimate based on the number of subscriptions to internal mailing lists for MBT would count only about 5-10% of product teams which are using or have tried using MBT for their daily tasks. While these numbers can be considered a success compared to other formal quality assurance approaches like verification, they are certainly not indicating a break-through. So what are the obstacles in applying MBT, and how can a larger group of users be attracted to the technology?

This paper first makes an attempt to answer this question, based on feedback from the user base of the Spec Explorer tool [5], its predecessor AsmL-T [3], and other internal MBT tools at Microsoft. The major issues, apart of the ubiquitous problem in the industry that people do not have enough time to try out new technology and educate themselves, seem to be the steep learning curve for modeling notations together with the lack of state-of-the-art authoring environments, missing support for scenario-based (interaction-based) modeling, thus involving not only the test organization but also other

K. Havelund et al. (Eds.): FATES/RV 2006, LNCS 4262, pp. 1–19, 2006.

stakeholders in the process, poor documentation of the MBT tools, and last but not least technical problems like dealing with state explosion, fine-grained test selection, and integration with test management tools.

The paper then sketches a new model-based testing environment which is currently under development at Microsoft Research and attempts to overcome some of the obstacles. The environment, called "Spec Explorer for Visual Studio" (for short, $\mathsf{SE}_{\mathsf{VS}}$), tries to address the identified challenges by providing a full integration into the development environment of Visual Studio, using a *multi-paradigmatic* approach to modeling, allowing to describe models on different levels of abstraction, using scenario and state oriented paradigms as well as diagrammatic and programmatic notations, and enabling the combination of those diverse artifacts for a given modeling and testing problem.

$\mathsf{SE}_{\mathsf{VS}}$ is internally based on the framework of action machines [6,7], which allows for uniform encoding of models which can stem from a variety of notations, and to combine and relate them using various compositions. The action machine framework supports the representation of models with symbolic parts in states and actions, which gives rise to the expressive power of defining partial models on a higher level of abstraction and compose them with lower-level models.

This paper is organized as follows. Sect. 2 describes lessons learned in applying MBT at Microsoft and draws some conclusions. Sect. 3 gives a high-level overview on the approach of the $\mathsf{SE}_{\mathsf{VS}}$ tool using examples. Sect. 4 gives a summary of the formalization of the underlying concepts, and Sect. 5 concludes.

2 Model-Based Testing in Practice: Lessons Learned

MBT has a long application tradition at Microsoft, and various tools have been and are in use. The first tool, the Test Modeling Toolkit (TMT), was deployed in 1999, and is based on extended finite state machines (EFSM) [1]. Microsoft Research deployed two tools, AsmL-T in 2002 [3] and Spec Explorer in 2004 [5], both using executable specification languages based on the the abstract state machine paradigm (ASM) [8] as the modeling notation. Other internal tools which have not been published are also around. The general mailing alias used for internal discussion of MBT issues at Microsoft currently exceeds 700 members.

All these tools, though quite different in details and expressiveness, share some common principles. Models are described by *guarded-update rules* on a global data state. The rules describe transition between data states and are labeled with *actions* which correspond to invocations of methods in a test harness or in the actual system-under-test (SUT). Rules can be parameterized (and the parameters then usually also occur in the action labels). A user provides value domains for the parameters, using techniques like pairwise combination or partitioning. In the approach as realized by AsmL-T and Spec Explorer, the parameter domains are defined by expressions over the model state, such that for example they can enumerate the dynamically created instances of an object type in the state where the rule is applied.

A very simple example to demonstrate the basic concepts as they appear in Spec Explorer today is considered. The model describes the *publish-subscribe* design pattern which is commonly used in object-oriented software systems. According to this pattern,

```
class Publisher {
  Set<Subscriber> subscribers = Set{};
  [Action(ActionKind.Controllable)]
  Publisher(){}
  [Action(ActionKind.Controllable)]
  void Publish(object data)
  {
    foreach (Subscriber sub
                    in subscribers)
      sub.mbox += Seq{data};
  }
}
```

```
class Subscriber {
  Seq<object> mbox = Seq{};
  [Action(ActionKind.Controllable)]
  Subscriber(Publisher publisher)
  {
    publisher.subscribers += Set{this};
  }
  [Action(ActionKind.Observable)]
  void Handle(object data)
  requires mbox.Count > 0 &&
           mbox.Head.Equals(data);
  {
    mbox = mbox.Tail;
  }
}
```

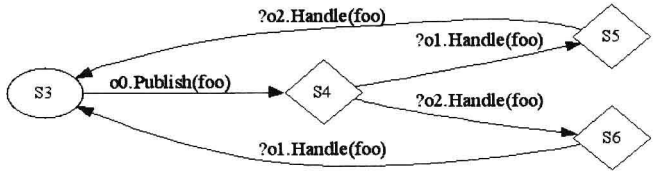

Fig. 1. Publisher-Subscriber Model

various subscriber objects are registered with a publisher object to receive asynchronous notification callbacks when information is published via the publisher object (in fact, the subscribers can dynamically register and unregister at a publisher, but this aspect is simplified here.) Thus this example includes dynamic object creation as well as reactive behavior.

The model is given in Fig. 1 (top). The state of the model consists of publisher and subscriber instances. A publisher has a field containing the set of registered subscribers, and a subscriber has a field representing the sequence of data it has received but not yet handled (its "mailbox"). The model simply describes how data is published by delivering it to the mailboxes of subscribers, and how it is consumed by a subscriber in the order it was published. The precondition of the handling method of the subscriber enables it only if the mailbox is not empty, and if the data parameter equals to the first value in the mailbox. Note that the `Handle` method is an *observable* action, which comes out as spontaneous output from the system under test (SUT).

Fig 1 (bottom) shows an excerpt from the state graph generated by Spec Explorer from this model. This kind of graph corresponds to an *interface automaton* [9]. In this fragment, one publisher and two subscribers are configured (the state graph omits the configuration phase). From state `S3`, a `Publish` invocation is fired, leading to state `S4`, which is an *observation* state where the outgoing transitions are observable actions. The meaning of an observation state is that the SUT has an internal choice to do *one* of the outgoing transitions, as opposed to a control state (`S3`) where it must accept *all* of the outgoing transitions. Thus, effectively, the model gives freedom to an implementation to process the subscribers of a publisher in any given order.

In order to generate the state graph, the model was augmented with further information: the parameters passed to the `Publish` method have been specified (here, `"foo"`), the number of publishers and subscribers to be created has been bounded, as well as the number of messages in the mailbox of a subscriber.

Such state graphs are then input to traversal algorithms to generate a test suite which can be executed offline, or are dynamically traversed using online/on-the-fly testing. For both cases, the test execution environment takes care of binding object values in the model to objects in the implementation, as well as queuing asynchronous observable action invocation events in the SUT for consumption by the model. For details, see [5].

In practice, models written with Spec Explorer are significantly larger than this simple example; yet they are rarely on the scale of real programs. In the applications at Microsoft, models typically introduce about 10 to 30 actions, with up to 2000 lines of model code, in exceptions over 10000 lines. Yet, these models are able to test features with a code-base which is larger by an order of magnitude or more. This stems from the level of abstraction chosen in modeling. Model-based testing is used for a wide range of application types, including user interfaces, protocols, windows core components, frameworks, device drivers, and hardware abstraction layers.

While in general successfully used in practice, the technology of Spec Explorer, as well of the other available tools at Microsoft, raises some challenges which hinder wider adoption. These will be discussed in the remainder of this section.

2.1 The Modeling Problem

Authoring. Computer folklore says: "every editor is better than a new editor". Though clearly this statement is sarcastic, one should not underestimate its wisdom. The author of this paper, for example, used to apply the `vi` editor (a great relict of very early Unix times) for his programming over many years, even though on the development platform Visual Studio was available, which provides automatic indentation, incremental compilation, context-sensitive completion, refactoring, and many more nice features.

When initially rolling out one approaches' favorite modeling notation to end users, the gravity of habits is even stronger: users are asked to use a new language in an authoring environment which usually does *not* provide the convenience features they are acquainted with from modern environments.

Notations have perhaps become less important today than the environments which support them. This at least applies to users which are heavily using these modern development environments – among which are most younger testers and developers. It might apply less to other stakeholders (like the author of this text, which is still using a `vi` emulation mode under Emacs to write this document in LaTeX).

The lesson learned is that if one comes up with a new notation, one should better be sure that either the users of that notation do not care about modern authoring support, or one should match this support. The later is unfortunately not trivial. The effort for decent authoring support for a language is probably an order of magnitude higher than providing its compiler.

Executable Specifications vs Programming Languages. The first generation of the Microsoft Research MBT tools was based on the Abstract State Machine Language

(AsmL), a high-level executable specification language, which integrates concepts from functional programming languages and specification languages like Z, B and VDM. Though the basic concepts of this language seem to be simple and intuitive (it uses a "pseudo-code" style notation and avoids any mathematical symbols), apart of some stellar exceptions, for most testers the learning curve was too steep (see [4] for a discussion).

Testers struggled with concepts like universal and existential quantification and set comprehensions. Under the assumption that the problem was not the concept itself but perhaps the unfamiliar way in which it was presented, the next generation, Spec Explorer, offered in addition to AsmL the Spec# notation, which disguised the high-level concepts in C# concrete syntax. Though this approach was more successful, the basic problems remained. Typically, beginners and even intermediate levels in Spec# prefer to write a loop where a comprehension would be much more natural and concise.

This phenomena is not just explained by the inability of users. It is more the *unwillingness* to learn many new concepts at the same time, in particular if they are not obviously coherent. Confronted with a new technology like MBT and the challenges to understand the difference between model and implementation and finding the right abstraction levels, having in *addition* the challenge to learn a new language, is mastered only by a minority.

Some people argue that a high-level notation which differs from the programming notations might support identifying different levels of abstractions, as they are essential for modeling. The AsmL and Spec# experiences do not confirm this, at least in the beginning of the adoption process. Rather, it seems that if the notation is mastered after some time, a misleading conceptualization takes place: abstraction is identified with notation, which after all is only syntactic sugar (in the case of executable specification languages). Someone who already masters the abstraction process will certainly benefit from a more concise way to write things down. But for others, the notation can be just a further roadblock in mastering the technology.

The conceptual distance between programming languages like C# and executable specification languages like Spec# is shrinking steadily. The new forthcoming C# version 3.0 will contain – in addition to the relatively declarative notational features C# has already now – support for comprehension notations (as part of the LINQ project [10]). When new language concepts are build into main-stream programming languages like C# or Java, a campaign is kicked off. Manufactures provide early technology previews, blogs and message boards are filled, books are written or newly edited, and so on. After some time, the concepts which might have appeared strange to the average programmer are familiar to many. Why trying to compete with this?

The lesson learned here is that it appears wiser not to mix evangelizing executable specification languages with the very core of model-based testing concepts. This should not mean that those notations do not have a place in MBT – they are indeed rather important. It just means that users should not be *forced* to use a new notation and environment in order to write their first models. Let them use existing programming notations and their authoring environments if they like. The core of a model-based testing approach and tool should be agnostic about this choice; it should be *multi-paradigmatic*.

Scaling Up to Model-Based Development. One of the promises of MBT is to be an entry door for model-based development. In course of applying MBT at Microsoft, several test teams have attempted to incorporate program managers, domain experts, business analysts, and the like into the modeling process. This has not been very successful so far, though some exceptions exist.

One interesting observation is that executable specification languages like AsmL, which provide a high-level pseudo-code style notation, are more attractive to those stakeholders than programming-oriented notations like Spec#. AsmL had more users authoring system models, compared to just models for test, whereas with the introduction of Spec# and Spec Explorer, these applications diminished. This is a strong argument to *continue* supporting high-level executable specification languages like AsmL for MBT (just do not make them the only choice).

However, it seems that the main obstacle here is not the language but the modeling style. AsmL, Spec#, or any of the other MBT approaches used at Microsoft are not attractive in the requirements phase since they are *state-based* instead of *scenario-based*. In this way they represent a design by itself – even if on an higher-level of abstraction. These high-level designs are well suited for analysis, but less well for understanding and communicating usage scenarios. Thus to incorporate stakeholders from the requirements league, scenario-based modeling must be supported.

Scenarios are also heavily used inside of the test organizations themselves. For example, *test plans* are commonly used at Microsoft to describe (in an informal way) what usage scenarios of a feature should be tested. These test plans, as well as the scenarios coming from the requirements phase, are intrinsically *partial*, omitting a lot of details, in particular oracles, parameter assignments, and so on. It is the job of the test engineers to "implement" these test plans.

The challenge for MBT to scale up to model-based development is the support of both the state-based and the scenario-based paradigm in one approach, where it is possible to *combine* (compose) models coming from those different sources. For example, a scenario might provide the control flow, and a state machine the oracle, and the composition of both produces an instantiated test suite.

How should scenario-based models be written down? In [11], a programmatic approach based on Spec# is suggested. While this approach is useful in some instances, diagrammatic approaches like activity charts or interaction charts look more promising, as far as stakeholders from the requirements phase should be involved. Because of the wealth of literature available, it seems wise to orient toward UML 2.0 when supporting diagrammatic notations, instead of inventing ones own. But again, the choice of the notation should *not* be part of the core of an MBT approach and tool.

Education and Documentation. For more than a decade, proponents of formal methods claim that the major problem in adoption is education. In particular universities are in charge of providing better preparation for those technologies. However, as long as there are no practical applications and tools around, only a minority of students will subscribe to this content.

Until then, the adoption problem must be solved in the field. To that end management support is the essence. At Microsoft, the most successful applications of MBT emerged in areas where the technology was pushed from management level by making

time resources available for the adoption phase. This has to go in combination with introduction classes and seminars, and – most important – good documentation and samples. See [4] for a discussion.

2.2 The Technology Problems

State Explosion. MBT is known to easily generate a huge amount of tests from even small models. But this turns out to be more a problem in practice than an advantage, commonly referred to as the "state explosion problem". In fact, this is the main concern mentioned by users of MBT tools at Microsoft.

The state explosion problem has a number of facets. First, the time required to run a test-suite is a significant cost factor. For example, at Microsoft, developers need to run so-called "basic verification tests" (BVT) before they can submit sources to a shared depot. The time required to run the BVT is important for development productivity. If BVTs require hours to finish, developers tend to submit their changes in larger time intervals, which raises problems with the integration of their changes with other developers changes.

This is also a reason why stochastic on-the-fly/online testing is not the solution for the state explosion problem. It is not realistic to run millions of tests "over night" in the standard development process. Indeed, this kind of testing has its proper use in test deployments which run in test labs asynchronously with the development process and in larger time intervals.

Test Selection. The notion of *test selection* is generally used in the MBT community to name the process of selecting some representative set of tests from the model. Thus it should provide the tools to overcome the state explosion problem. Test selection traditionally covers graph traversal techniques which can be applied to models which are boiled down to some finite state machine representation, as well as techniques for generating parameters of tested actions, like pairwise combination, partitioning, and so on. In the context of models which have an unbounded state space, like Spec Explorer models, test selection can also include bounds, filters, state grouping, and other techniques to prune the state space.

While these techniques are mostly automated and well understood, it is a regular complain of MBT users at Microsoft that they have not enough *fine-grained* control over the test selection process. For example, a typical user problem is to choose the set of tests from the model where during some initialization phase an arbitrary path is sufficient, in the operation phase paths should be chosen such that all transitions are covered, and in the shutdown phase again an arbitrary path is good enough. MBT tools need to support this kind of fine-grained control over the test selection process.

Some tools support defining so-called *test purposes* which are combined with the model to slice some desired behavior, using special notations for that [12,13]. Instead of introducing a further notation for describing test purposes, it looks desirable to use models to express test purposes and view the test selection problem with test purposes as a model composition problem. Test purposes then fall together with test plans and requirement scenarios, as discussed previously. Even more than for those applications, models used as test purposes must allow to express partial behavior which omits many details.