From the How to the What

Tiziana Margaria[1] and Bernhard Steffen[2]

[1]Universität Göttingen, Lotzestr. 16-18, 37083 Göttingen (Germany),
margaria@cs.uni-goettingen.de
[2]Universität Dortmund, Baroper Str. 301, 44221 Dortmund (Germany),
steffen@cs.uni-dortmund.de

Abstract. In this paper, we consider the Grand Challenge under a very specific perspective: the enabling of application experts without programming knowledge to reliably model their business processes/applications in a fashion that allows for a subsequent automatic realization on a given platform. This goal, which aims at simplifying the tasks of the many at the cost of ambitious and laborious tasks for the few, adds a new dimension to the techniques and concepts aimed at by the Grand Challenge: the application-specific design of platforms tailored for the intended goal. We are convinced that the outlined perspective provides a realistic and economically important milestone for the Grand Challenge.

1 Motivation

Since the very beginning of computer science, the mismatch between design for machines and design for brains has been a constant pain, and a constant driver for innovation: it was always clear that descriptions that are good for machine processing are inadequate for an intuitive human understanding, and that descriptions which are structured for easy comprehension contain a lot of 'syntactic overhead', that typically slows down their automatic processing.

Compilers were designed to overcome, or better to weaken, this mismatch: Rather than trying to construct machines that work as humans think, it seemed more appropriate to translate comprehensible descriptions into machine-adequate representations. Besides classical compilers that translate high-level programming language into machine code there are also other means of automated code generation from more abstract descriptions: to this group belong parser generators, data flow analysis generators, compiler generators, and all the modern OO-tools that translate, e.g., UML descriptions into code fragments. Most drastic are those versions of Model Driven Design (MDD) that aim at totally replacing the need of programming for most application development using model construction. Thus the original desire to lift low-level machine code (prescribing **How** to run the machine) to more abstract programs (describing **What** to achieve) has become reality at increasing levels of abstraction from the hardware, and leading to extreme approaches like requirements-based programming, which aims at automatically making the user's or customer's requirements executable.

2 Background

The fact that writing a parser, or even a compiler (which has been a real enterprise in the '60s and early '70s) has become meanwhile almost a trivial exercise, and, perhaps even more surprisingly, the fact that global heterogeneous software systems comprising several hundreds of millions of lines of code do quite a reliable service, certainly bases on conceptual advances. In particular techniques like type checking have had a strong impact on theory and practice: they filter out e.g. typos extremely efficiently and user friendly, and at the same time they guarantee an abstract notion of consistency. However, in our opinion, the current reliability of huge systems that run on heterogeneous platforms and that are increasingly based on third party legacy components, like middleware, container software, security protocols etc., is due to other, less scientific reasons:

1. Extensive use of *exception mechanisms*, especially in the presence of third party legacy components. In particular the code for internet applications often comprises more than 80% of such non-functional code in order to deal with the uncertainties of the net-based context.
2. *Pattern-based* programming styles, which do not focus on conciseness or efficiency of the code but on its comprehensibility.
3. Syntax-based *tool support*. Of course, semantics-based tools have a much higher potential, but, except for some niche success stories like the SLAM-project [1, 2], they did not really reach so far the broad industrial practice.
4. *Testing* and post-release *debugging* by a huge user community.

These four mechanisms have a huge combined pragmatic impact, which is probably the reason for the absence of the 'software crisis' forecasted long ago. However, they are in our opinion not a real safeguard: ultimately, they do not scale. They carry a long way, but they are deemed to fail in the long run as complex software is inherently meta-stable. A tested system may run for years, apparently very reliably, until some apparently minor modification leads to some totally uncontrollable effects. We all experienced situations where such minor modifications to systems considered stable suddenly revealed some deeper hidden bugs, which then took an enormous effort to repair. The larger the systems, the more distributed the features, the larger the project teams, the more likely is the long-term meta-stability of the product. The reason for this is that systems evolving over the years escape more and more our comprehension and thus our prevention and control. Late discovered errors may be so fatal that it is almost impossible to repair the system. Thus they are business-critical for organizations that produce and use those systems.

3 Our Proposal: Application-Specific Solutions

We believe that there is no general cure to this problem. Rather, we are convinced that we can only maintain control of complex systems if we develop tailored solutions for application-specific scenarios, since only for very restrictive settings

- based on clear patterns with predefined meaning - we will be able to capture enough semantics to regain sufficient control. This is important since with the variety of systems, platforms, technologies, communication and execution paradigms comes the need to bridge all this diversity in a harmonic way.

Our approach, called Aggressive Model Driven Design (AMDD), aims at decomposing the problem of compatibility and consistency of software (mass-) construction and customization into a

- (global) application-level problem, where compatibility and consistency is not a platform issue, but an issue of application/business process modelling, and therefore a matter of a considerably higher level of abstraction, and a
- platform-oriented synthesis/compilation issue, which provides the adequate 'technology mapping' that bridges the gap from the modelling-level over the programming language level to the target code.

The ultimate goal of this approach is to enable application experts without programming knowledge to reliably model their business processes in a fashion that allows for a subsequent automatic realization on a given platform. Thus, in contrast to usual compiler scenarios, which are designed to economically cover a wide range of source languages and target machines, our approach aims at automatically putting application/business-level models into operation. This is valuable and appreciated even at the price of a very expensive source/target mapping for each individual application/platform pair, and comes together with specific restrictions (e.g to respect standards and platform constraints) and supported patterns. This approach of domain/application-specific solutions goes far beyond established approaches based on domain-specific languages, as it essentially sees the role of the IT as a domain/application-specific platform *provider*. In fact, with AMDD the IT people (in contrast with the business/application experts), are active mainly *prior* to the application/process development to:

- support the domain modelling,
- design the most relevant patterns,
- provide the vital analysis, synthesis and compilation methods, and
- define constraints that guide the pattern-based application development in a way that guarantees the applicability of the provided methods and the reliability of the applications when running on their dedicated platform.

4 Grand Challenge: The AMDD-Persective

We are convinced that products that provide such application/platform-pairs for specific business solutions will have an enormous market already in the near future, as many current businesses complain about the delays imposed by the usual 'loop' via the IT section, even when the current business process only needs some minor modification.

On the other hand, this new scenario also provides a very interesting opportunity for the compiler community, as it allows one to set the scene in such a way

that powerful analysis and verification methods work. In fact, rather than being confronted with a fixed scenario, typically a programming language and a class of architectures, and with the question, what can be done under these circumstance, the new setting is goal oriented: We must provide a scenario, where the application expert is able to reliably control his business processes by himself.

As indicated in the previous section, this comprises not only the development of powerful synthesis and compilation techniques, but also

- the development of adequate domain modelling and design patterns (design for verification/synthesis/compilation), and
- the definition of (constraint-based) guiding mechanisms for the application expert that guarantee the safe deployment of the developed applications on its dedicated platform. In particular, this comprises a new quality of diagnostic information and feedback: it is the application expert who must be able to correct his processes in case of problems.

The applicability range of this perspective of the Grand Challenge is, of course, quite restrictive, as it requires the development of very specific setups. On the other hand, it goes beyond the vision of a 'Verifying Compiler', as it explicitly includes the definition of adequate setups, and significantly extends the corresponding requirements.

The next section will briefly sketch a framework and some, admittedly still rather restrictive, scenarios where this goal has been achieved: in the definition of Value-Added Telecommunication Services, for Test Management of Computer/Telephony Applications, and for Personalized Online Services.

5 AMDD and the ABC Framework

AMDD aims at the top-down organization of Service interoperation, which moves most of the recurring problems of compatibility and consistency of software (mass) construction and customization from the coding and integration level to the modelling level (see Fig. 1). Rather than using component models - as usual in today's Component Based Development paradigm - just as a means of specification, which

- need to be compiled to become a 'real thing' (e.g., a component of a software library),
- must be updated (but typically are not), whenever the real thing changes
- typically only provide a local view of a portion or an aspect of a system,

we use *application models* (called Service Logic Graphs (SLGs) in the ABC). Application models are the center of the design activity, becoming *the* first class entities of the *global* system design process. In such an approach, as briefly sketched in Sect. 5.1,

- *libraries* are established at the modelling level: building blocks are (elementary) models, rather than software components,

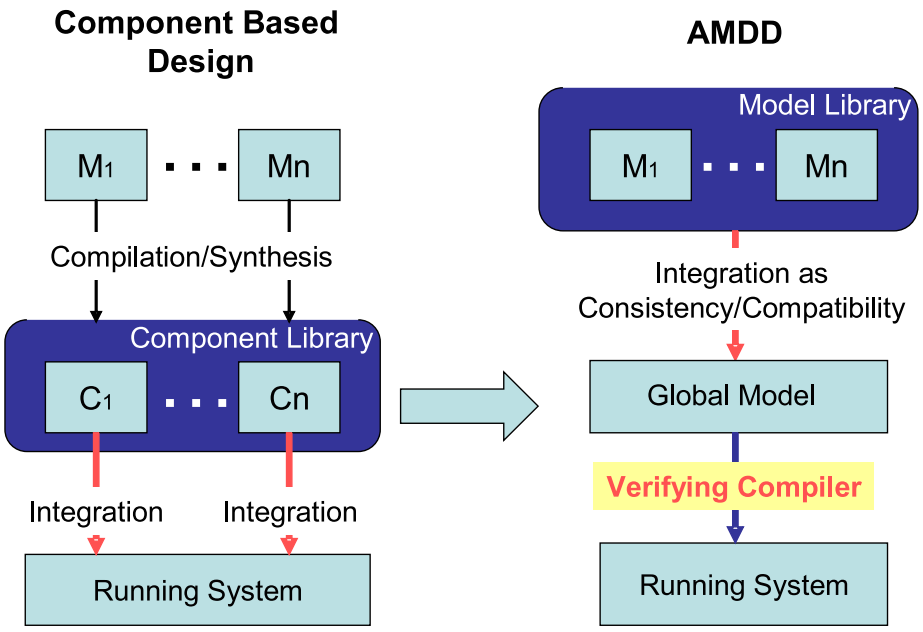

Fig. 1. AMDD: From the How to the What

- *applications* are specified by model combinations (composition, configuration, superposition, conjunction...), viewed as a set of constraints that the implementation needs to satisfy,
- global model combinations are *compiled* (or synthesized, e.g. by solving all the imposed constraints) into a homogeneous solution for a desired platform,
- *application changes* (upgrades, customer-specific adaptations, new versions, etc.) happen primarily (and ideally, only) at the modelling level, with a subsequent global recompilation or re-synthesis.

This *aggressive* style of *model-driven development* [12] strictly separates compatibility and migration issues from model/functionality composition and heavily relies on compilable and synthesisable models. This way it is possible to better control or even in some cases overcome the problem of incompatibility between (global) models, (global) implementations, and their components, because, due to the chosen setups, compatibility can be established and guaranteed at development time, and later-on maintained by (semi-) automatic compilation and synthesis.

In fact, AMDD, when applicable, has the potential to drastically reduce the long-term costs due to version incompatibility, system migration and upgrading. Thus it helps protecting the investment in the software infrastructure. We are therefore convinced that this aggressive style of model-driven development will become the development style at least for mass-customized software in the future.

5.1 An AMDD-Realization

The Application Building Center (ABC) developed at METAFrame Technologies in cooperation with the University of Dortmund promotes the AMDD-style of development in order to move the application development for certain classes of applications towards the application expert. The ABC has been successfully used in the design and customization of Intelligent Network Services [8], test environments [7, 11], distributed online decision support systems [5, 4], and Web Services [9], and it is currently being used in project for Enterprise Resource Planning (ERP) and Supply Chain Management (SCM).

Even though the ABC should only be regarded as a first step of AMDD approach, it comprises the essential points of AMDD, that concern dealing with a heterogeneous landscape of models, supporting a number of formal methods, providing tools that enforce formal-methods based validation and verification, and providing automatic deployment and maintenance support [9]. The rest of the paper will briefly sketch some running application scenarios.

6 Running Scenarios

This section presents three practically relevant applications, ranging from an enhancement of the workflow capabilities of a content management system for non-experts, over our Integrated Test Environment, which enables test experts without deep programming knowledge to create and modify their testing scenarios, to the Online Conference Service, a rather complex distributed, role-based, and personalized online decision support system.

6.1 Enhancing Workflows of a Content Management System

In this project, we used the restrictive workflow management functionalities of a commercial content management system (CMS) as a basis for a component library in the ABC, added global features, like e.g. a version management functionality, and taxonomically classified the resulting library of functionalities. This directly enabled us to graphically design workflows far beyond the capabilities of the original CMS and to embed them in other projects.

Besides increasing the modelling power and the range of applicability, using the ABC also allowed us to check the consistency of the workflows. A simple but important constraint we checked was that *'a new page will never be published before it is approved'*. After a simple translation into temporal logic, this constraint can now automatically be model checked for any of the workflows within a small fraction of a second. It turned out that already this constraint did not hold for quite some standard workflows of the CMS.

Thus, using the model checking feature of the ABC, it was straightforward to enhance the CMS environment to avoid such mistakes once and forever, and to combine the CMS features for an editorial workflow with additional features like version control, automated update cycles, and features for fault tolerance, e.g. for taking care of holidays or illness during the distribution of labor.

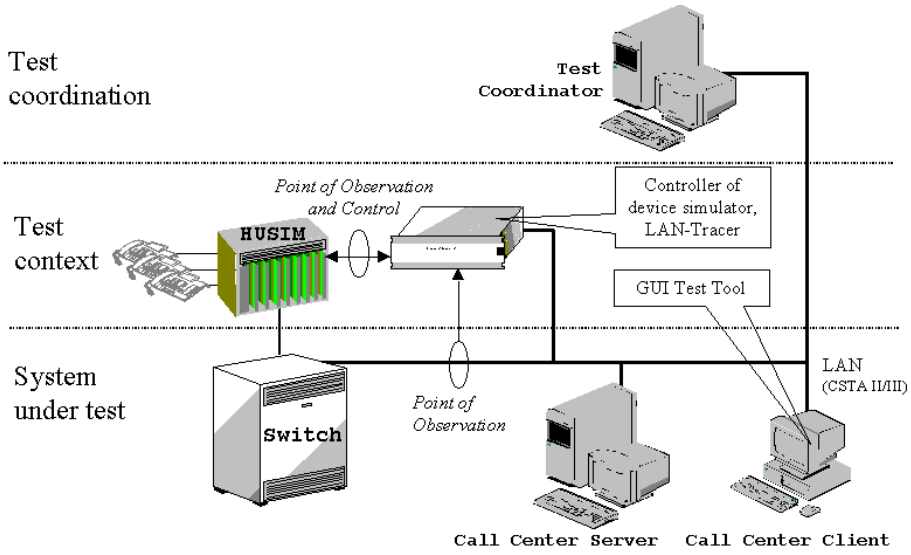

Fig. 2. Architecture of the Test Setting for the CTI Application

6.2 ITE: The Integrated Test Environment

A completely different application is the Integrated Testing Environment (ITE) for system level test of complex distributed systems [13, 7] developed in a project with Siemens ICN in Witten (Germany). The core of the ITE is the test coordinator, an independent system that drives the generation, execution, evaluation and management of the system-level tests. In general, it has access to all the involved subsystems and can manage the test execution through a coordination of different, heterogeneous test tools. These test tools, which locally monitor and steer the behavior of the software on the different clients/servers, are technically treated just as additional units under test, which led to the system depicted in Fig. 2. The ITE has been successfully applied along real-life examples of IP-based and telecommunication-based solutions: the test of a web-based application (the Online Conference Service described below [14]) and the test of IP-based telephony scenarios (e.g. Siemens' testing of the Deutsche Telekom's Personal Call Manager application [7], which supports among other features the role based, web-based reconfiguration of virtual switches).

In this project we practiced the AMDD approach at two levels:

- the modelling of the test environment itself, and
- the modelling of test cases.

The benefits of the AMDD approach became apparent once a drastic change of the requirements of the test scenario in the telephony application occurred, which meant a new quality of complexity along three dimensions ([7]):

– testing over the internet,
– testing virtual clusters, and
– testing a controlling system in a non-steady state (during reconfiguration).

We could inherit a lot of the conceptual structure of the 'old' ITE for the new version of the test environment. Even more striking was the fact that the test cases hardly needed any adaption, except for some specific changes directly related to the functionality changes. Thus a change that Siemens considered to be 'close to impossible' became a matter of a few weeks [12].

6.3 OCS: The Online Conference Service

The OCS (Online Conference Service) is a server-based Java application that customizes a heavily workflow-oriented application built with the ABC [5, 4, 6]. It proactively helps authors, Program Committee chairs, Program Committee members, and reviewers to cooperate efficiently during their collaborative handling of the composition of a conference program. The service provides a timely, transparent, and secure handling of the papers and of the related tasks for submission, review, report and decision management. Several security and confidentiality precautions have been taken, in order to ensure proper handling of privacy and of intellectual property sensitive information. In particular,

– the service can be accessed only by registered users,
– users can freely register only for the role Author,
– the roles Reviewer, PC Member, and PC Chair are sensitive and conferred to users by the administrator only,
– users in sensitive roles are granted well-defined access rights to paper information,
– users in sensitive roles agree to treat all data they access within the service as confidential.

The service has been successfully used for over 50 computer science conferences, including the full ETAPS Conferences (with 5 instances of the OCS running in parallel). The Online Conference Service allows fully customizable, role-based business-process definitions, it is tailored for personalized support of each participant in the course of the operations of a virtual Program Committee meeting, and it is customizable and flexibly reconfigurable online at any time for each role, for each conference, and for each user [3].

The AMDD approach drastically simplified the realization and organization of the steady evolution of the OCS, which was guided by the growing demands of the users. It allowed to completely separate the issues of functionality implementation from the workflow (process) modelling (in term of SLG's), to reuse in particular the constraints for the permission handling. In fact, this property remained even true when developing an Online Journal Service (OJS), which required to change most of the workflows, and the addition of new functionality.

7 Conclusions and Perspectives

We have presented our favorite perspective of the Grand Challenge: the technical realization of Aggressive Model Driven Design (AMDD). This concerns enabling application experts without programming knowledge to reliably model their business processes in a fashion that allows for a subsequent automatic realization on a given platform. In contrast to usual compiler scenarios, which are designed to economically cover a wide range of source languages and target machines, this requires the dedicated treatment of each individual application/platform-pair together with its specific restrictions and supported patterns. This approach of domain/application-specific solutions goes far beyond established approaches based on domain-specific languages, as it essentially sees the role of the IT as a domain/application-specific platform provider, active mainly *prior* to the application/process development.

In order to achieve this goal, which makes application development *difficult for the few* (the providers of domain-specific platforms), but *easy for the many* (the application experts), the domain-specific platform must enforce all the necessary constraints necessary for a safe deployment of the designed processes/applications, and it must give application-level feedback to the process designers. This is only possible on the basis of very strong analysis and verification techniques, specifically adapted and applied to the domain-specific scenarios.

Of course, AMDD will never replace genuine application development, as it assumes techniques to be able to solve problems (like synthesis or technology mapping) which are undecidable in general. On the other hand, more than 90% of the application development costs arise worldwide at a rather primitive development level, during routine application programming or software update, where there are no technological or design challenges. There, the major problem faced is software quantity rather than achievement of very high conceptual complexity, and automation should be largely possible. AMDD is intended to address (a significant part of) this 90% 'niche', which we consider a particularly interesting and realistic scenario also for the Grand Challenge.

References

1. T. Ball, S. Rajamani: *Automatically Validating Temporal Safety Properties of Interfaces*, SPIN 2001, Workshop on Model Checking of Software, LNCS 2057, May 2001, pp. 103-122.
2. T. Ball, S. Rajamani: *Debugging System Software via Static Analysis*, POPL 2002, January 2002, pages1-3.
3. M. Karusseit, T. Margaria: *Feature-based Modelling of a Complex, Online-Reconfigurable Decision Support Service* WWV'05, 1st Int'l Workshop on Automated Specification and Verification of Web Sites, Valencia, Spain, March 14-15, 2005, – final version appears in ENTCS.
4. B. Lindner, T. Margaria, B. Steffen: *Ein personalisierter Internetdienst für wissenschaftliche Begutachtungs- prozesse* - In Proc. GI-VOI-BITKOM- OCG-

TeleTrusT Konferenz on Elektronische Geschäftsprozesse (eBusi- ness Processes), Universität Klagenfurt, Sept. 2001, http://syssec.uni-klu.ac.at/EBP2001/ .

5. T. Margaria: *Components, Features, and Agents in the ABC*, invited contribution to the volume *Components, Features, and Agents*, PostWorkshop Proceedings of the Dagstuhl Seminar on *Objects, Agents and Features* 7-21.3.2003, H.-D. Ehrich, J.-J. Meyer, and M. Ryan eds., appears in LNCS, Springer Verlag.
6. T. Margaria, M. Karusseit: *Community Usage of the Online Conference Service: an Experience Report from three CS Conferences*, 2nd IFIP Conference on "e- commerce, e-business, e-government" (I3E 2002), Lisboa (P), 7-9 Oct. 2002, in "Towards the Knowledge Society - eCommerce, eBusiness and eGovernment", Kluwer Academic Publishers, pp.497-511.
7. T. Margaria, O. Niese, B. Steffen, A. Erochok: *System Level Testing of Virtual Switch (Re-)Configuration over IP*, Proc. IEEE European Test Workshop, Corfu (GR), May 2002, IEEE Society Press.
8. T. Margaria, B. Steffen: *METAFrame in Practice: Design of Intelligent Network Services*, in "Correct System Design - Issues, Methods and Perspectives", LNCS 1710, Springer-Verlag, 1999, pp. 390-415.
9. T.Margaria, B. Steffen: *Second-Order Semantic Web* Proc. SEW-29, 29th Annual IEEE/NASA Software Engineering Workshop, April 2005, Greenbelt (USA)., IEEE Computer Soc. Press.
10. T. Margaria, B. Steffen: *Backtracking-free Design Planning by Automatic Synthesis in METAFrame.* Proc. of Int. Conf. on Fundamental Aspects of Software Engineering (FASE'98), Lisbon, Portugal. (eds.: E. Astesiano), Lecture Notes in Computer Science (LNCS), Vol. 1382, pp. 188–204. Springer-Verlag, Heidelberg, Germany, March 30 - April 3 1998.
11. T. Margaria, B. Steffen: *Lightweight Coarse-grained Coordination: A Scalable System-Level Approach*, to appear in STTT, Int. Journal on Software Tools for Technology Transfer, Springer-Verlag, 2003.
12. T. Margaria, B. Steffen: *Aggressive Model Driven Development of Broadband Applications.* invited contribution for the book: Delivering Broadband Applications: A Comprehensive Report, IEC, Int. Engineering Consortium, Chicago (USA), 2004.
13. O. Niese, T. Margaria, A. Hagerer, M. Nagelmann, B. Steffen, G. Brune, H. Ide: *An automated testing environment for CTI systems using concepts for specification and verification of workflows. Annual Review of Communication*, Int. Engineering Consortium Chicago (USA), Vol. 54, pp. 927-936, IEC, 2001.
14. O.Niese, T. Margaria, B. Steffen: Demonstration of an Automated Integrated Test Environment for Web-based Applications, - in "Model Checking of Software", Proc. SPIN 2002, 9th Int. SPIN Workshop on Applications of Model Checking, satellite Workshop to ETAPS 2002, Grenoble (F), April 2002, LNCS N. 2318, pp. 250-253, Springer Verlag.