Model-based integration and testing of high-tech multi-disciplinary systems

Model-based Integration and Testing of High-tech Multi-disciplinary Systems

Niels Cornelis Wilhelmus Maria Braspenning

Voorkant: De voorkant van dit proefschrift illustreert de theorie en de praktijk van model-gebaseerd integreren en testen. Onderaan wordt een theoretisch proces voor systeemontwikkeling getoond, waarin een model M een realisatie Z kan vervangen voor vroegtijdige integratie en systeem tests. Bovenaan wordt een praktische toepassing van dit proces voor de EUV wafer scanner van ASML getoond, waarbij begonnen wordt met alleen modellen en waarbij deze stapsgewijs door realisaties worden vervangen. De kleurgradiënt in de achtergrond beeldt een van de doelen van dit proefschrift uit, namelijk het overbruggen van de kloof tussen theorie en praktijk.

Cover: The cover of this thesis illustrates the theory and practice of model-based integration and testing. At the bottom, a theoretical system development process is shown, in which a model M can replace a realization Z for early integration and system testing. At the top, a practical application of this process to the EUV wafer scanner of ASML is shown, starting with models only and gradually replacing them by realizations. The color gradient in the background represents one of the goals of this thesis, i.e., bridging the gap between theory and practice.

Cover design: Niels Braspenning

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics). IPA Dissertation Series 2008-05.

A catalogue record is available from the Eindhoven University of Technology Library.

ISBN: 978-90-386-1204-1

Reproduction: Universiteitsdrukkerij Technische Universiteit Eindhoven

The work described in this thesis has been carried out as part of the TANGRAM project under the responsibility of the Embedded Systems Institute, partially supported by the Netherlands Ministry of Economic Affairs under grant TSIT2026. The work has been carried out at ASML Veldhoven and at the Eindhoven University of Technology, both in the Netherlands.

Model-based Integration and Testing of High-tech Multi-disciplinary Systems

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van
de Rector Magnificus, prof.dr.ir. C.J. van Duijn,
voor een commissie aangewezen door het College
voor Promoties in het openbaar te verdedigen op
maandag 18 februari 2008 om 16.00 uur

door

Niels Cornelis Wilhelmus Maria Braspenning

geboren te Breda

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. J.E. Rooda
en
prof.dr. J.C.M. Baeten

Copromotor:
dr.ir. J.M. van de Mortel-Fronczak

Preface

In 2003, while I was performing my master's project as part of an industrial Ph.D. project, I became enthusiastic about performing academic research and using its results to solve real problems in industrial practice. At the same time, ASML and the Embedded Systems Institute (ESI) initiated the TANGRAM project to investigate methods, techniques, and tools for the integration and test problem as experienced by ASML. Now, four years later, I can say that the TANGRAM project provided me with the right context to put my enthusiasm into effect.

This thesis is the final result of my Ph.D. project and is titled 'Model-based Integration and Testing of High-tech Multi-disciplinary Systems'. Although this title refers to the integration and testing of high-tech multi-disciplinary systems, it could also have 'Integration and Testing of Academic Theory and Industrial Practice' as a subtitle. One of the interesting challenges of this Ph.D. project was to investigate how theoretical concepts from various academic fields related to model-based engineering can be 'integrated' in the current industrial practice of integration and testing at ASML, and to 'test' the practical applicability and profitability of this approach by applying it in real industrial case studies. I would like to thank various people who supported me during this integration and testing process.

First of all, I would like to thank professor Koos Rooda for offering me the opportunity to perform this Ph.D. project within the Systems Engineering Group and for his supervision during the project. I would also like to thank Asia van de Mortel-Fronczak for coaching me and for reviewing the many pages of text that I have produced the last few years. Furthermore, I thank my second promotor Jos Baeten and the other members of the reading committee, professor Arjan van Gemund, professor Fred van Keulen, and professor Maarten Steinbuch, for their valuable comments on my thesis. Special thanks goes to Johan Neerhof from ASML, for bringing his industrial experience and point of view into the mix during our weekly MBI&T meetings and for helping me with practical issues at ASML. I am very pleased to have Johan as advisor in my Ph.D. defence committee.

The TANGRAM project brought together an interesting mix of people from different backgrounds to work on the topic of integration and testing. I liked being part of that mix and would like to thank all TANGRAM project members, not only for their cooperation and fruitful discussions, but also for all the fun we had. Special thanks goes to Roel Boumen and Ivo

de Jong, who joined me on the journey through Ph.D. country and through other countries as well, I will never forget our adventures in China and the USA. I would like to thank the students who helped me during the Ph.D. project as part of their bachelor or master assignment: Rob Hendrikx, Kasia Peplowska, Jusuf Anggono, Gijs van Bokhoven, Ton Geubbels, and Esmée Bertens. I also thank the following people from ASML and ESI for contributing to my Ph.D. project by being advisor, co-author of a paper, or reviewer: Luud Engels, Tom Brugman, Tammo van den Berg, Durk van der Ploeg, and Herman Driessen from ASML and Jan Tretmans, Frank Pijpers, and Dragan Kostić from ESI. Finally, I would like to thank the ASML engineers for their time and support in the EUV case study, I think that we both learned many things from the experience in this case study.

I thank my colleagues at the Systems Engineering Group of the Eindhoven University of Technology for the pleasant working atmosphere. It is nice to see the MBI&T method appearing in other Systems Engineering papers and in course material for Systems Engineering students. A special word of gratitude goes to Albert Hofkamp, Ralph Meijer, and Henk van Rooy. Being one of the early adopters of the new Chi 1.0 toolset and the real-time, distributed simulator in particular, I asked them many questions and provided them with many bug reports and all kinds of feature requests. I think that without their help in solving these issues, the EUV case study would not have been so successful. Furthermore, I thank my Ph.D. colleagues Elena Bortnik and Roel Boumen for co-authoring two papers. Special thanks to Mieke Lousberg for taking care of many non-technical issues.

Finally, I would like to thank my family, relatives, and friends for their interest in my work and for helping me to think about other things than writing papers and a thesis. Last but certainly not least, I thank Maaike for all her love and support, and for the large amount of patience she had to show, especially in the last few months.

Zundert, December 2007

Summary

Model-based Integration and Testing of High-tech Multi-disciplinary Systems

In the current industrial practice of high-tech multi-disciplinary system development, it is difficult to deal with the many system development challenges due to system complexity, market pressure, and resource limitations. As a result, the effort required to integrate the components and to test the resulting system against the initial requirements increases significantly. The integration and test (I&T) phases have a large and growing influence on time-to-market and product quality, which are the main business drivers for developers of high-tech multi-disciplinary systems such as the wafer scanners from ASML.

The main objective of the Ph.D. project described in this thesis is to reduce time-to-market and to improve product quality by using models as replacements of not yet available component realizations for early integration and system testing. Early integration and system testing distributes the total I&T effort over a wider time frame, reducing the pressure on the final I&T phases. Furthermore, problems are detected at an earlier stage, which reduces the costs for fixing them and which improves product quality. Finally, the use of models enables the application of model-based techniques and tools for thorough system analysis and systematic testing, which help to improve the system overview for the engineers.

In the proposed model-based integration and testing (MBI&T) method, formal and executable models are used as early representations of not yet realized components that can be integrated with already realized components. Before component realizations are available, model-based analysis techniques such as simulation and model checking can be used to analyze whether the behavior of the system model corresponds to the intended behavior as specified in the system requirements and system design. When a component realization becomes available, it can be tested against the corresponding component model by automatic model-based testing. Using an appropriate integration infrastructure that implements the designed and modeled interaction behavior, the realized components can be integrated with the models of not yet realized components. The resulting model-based integrated system can then be tested on the system level before the complete system realization is available.

To show the practical applicability and the profitability of the MBI&T method, the method was instantiated with the process algebraic language χ (Chi) and its toolset and it was applied to an industrial case study concerning the component interaction and time behavior in an ASML wafer scanner. The results of this case study showed that the method can be applied in industrial practice, dealing with the (not always optimal) conditions and constraints of the

current way of working, e.g., incomplete specifications. In the case study, several system design and integration problems were detected and prevented at an early stage, which increased the system overview for the engineers and prevented significant amounts of expensive test time and rework. This shows that the MBI&T method contributes to a shorter time-to-market and to improved product quality.

In order to decide where and when the I&T phases can benefit from models, the costs of using models to improve time-to-market and product quality must be quantified, which is possible by using integration and test sequencing techniques. The proposed quantitative decision making process was applied to the I&T phases of a new wafer scanner, showing the possibility to determine where and when it is profitable to use models for integration and testing.

Samenvatting

Modelgebaseerd integreren en testen van high-tech multi-disciplinaire systemen

In de huidige industriële praktijk van het ontwikkelen van high-tech multi-disciplinaire systemen is het moeilijk om om te gaan met de vele uitdagingen van systeemontwikkeling, wat veroorzaakt wordt door de complexiteit van de systemen, door druk vanuit de markt en door beperkte middelen. Als een gevolg hiervan neemt de inspanning die nodig is om de componenten te integreren en om het resulterende systeem te testen tegen de initiële eisen significant toe. De integratie en test (I&T) fasen hebben een grote en toenemende invloed op de doorlooptijd en de kwaliteit van het product, welke de belangrijkste drijfveren zijn voor bedrijven die high-tech multi-disciplinaire systemen ontwikkelen zoals de wafer scanners van ASML.

Het hoofddoel van het promotieproject zoals beschreven in dit proefschrift is het reduceren van de doorlooptijd en het verbeteren van de kwaliteit van het product, dit door middel van modellen die gebruikt worden ter vervanging van nog niet beschikbare componentrealisaties om vroegtijdige integratie en systeemtests mogelijk te maken. Door vroegtijdige integratie en systeemtests wordt de totale I&T inspanning verdeeld over een grotere tijdspanne, waardoor de druk op de laatste I&T fasen afneemt. Daarnaast worden problemen in een eerder stadium gedetecteerd, wat de kosten om de problemen te verhelpen verlaagt en de kwaliteit van het product verhoogt. Tenslotte maken modellen het mogelijk om modelgebaseerde technieken en gereedschappen toe te passen voor grondige systeemanalyse en voor systematisch testen, welke de ingenieurs helpen om hun systeemoverzicht te verbeteren.

In de voorgestelde modelgebaseerde integratie en test (MBI&T) methode worden formele en executeerbare modellen gebruikt als vroegtijdige representaties van nog niet gerealiseerde componenten, welke geïntegreerd kunnen worden met reeds gerealiseerde componenten. Voordat er componentrealisaties beschikbaar zijn, kunnen modelgebaseerde analysetechnieken zoals simulatie en model checking gebruikt worden om te onderzoeken of het gedrag van het systeemmodel overeenkomt met het beoogde gedrag zoals gespecificeerd in de systeemeisen en in het systeemontwerp. Wanneer een componentrealisatie beschikbaar komt, kan deze getest worden ten opzichte van het betreffende componentmodel door middel van automatisch modelgebaseerd testen. Gebruikmakend van een geschikte integratie infrastructuur die het ontworpen en gemodelleerde interactiegedrag implementeert, kunnen de gerealiseerde componenten geïntegreerd worden met de modellen van nog niet gerealiseerde componenten. Het resulterende, op basis van modellen geïntegreerde systeem kan dan gebruikt

worden om op systeemniveau te testen voordat de complete systeemrealisatie beschikbaar is.

Om de praktische toepasbaarheid en het profijt van de MBI&T methode aan te tonen, is de methode geïnstantieerd met de procesalgebraïsche taal χ (Chi) en de bijbehorende gereedschappen en is de methode toegepast op een industriële casus met betrekking tot de component interactie en het tijdsgedrag in een ASML wafer scanner. De resultaten van deze casus laten zien dat de methode toegepast kan worden in de industriële praktijk, omgaand met de (niet altijd optimale) omstandigheden en randvoorwaarden van de huidige manier van werken, zoals bijvoorbeeld incomplete specificaties. In de casus werden meerdere systeemontwerp- en integratieproblemen in een vroegtijdig stadium gevonden en voorkomen, waardoor het systeemoverzicht voor de ingenieurs verbeterde en waardoor een significante hoeveelheid tijd voor testen en herzieningen bespaard werd. Dit laat zien dat de MBI&T methode bijdraagt aan een kortere doorlooptijd en aan een verbeterde kwaliteit van het product.

Om te kunnen besluiten waar en wanneer de I&T fasen kunnen profiteren van modellen, moeten de kosten van het gebruik van modellen om de doorlooptijd te verkorten en om de kwaliteit van het product te verbeteren gekwantificeerd worden, wat mogelijk is door gebruik te maken van technieken om integratie- en testvolgorden te bepalen. Het voorgestelde kwantitatieve besluitvormingsproces is toegepast op de I&T fasen van een nieuwe wafer scanner, wat de mogelijkheid laat zien om te bepalen waar en wanneer het winstgevend is om modellen te gebruiken om te integreren en te testen.

Contents

CHAPTER 1

Introduction

The 'why' and the 'what'...

This thesis is the final result of a Ph.D. project on 'Model-based Integration and Testing of High-tech Multi-disciplinary Systems', part of a larger research project titled 'Test approach based on integrated product generation and product realization applied to ASML machines' or TANGRAM for short [TANGRAM project 2007; Tretmans 2007]. This chapter describes the background, the problem context, and the objectives of the Ph.D. project, as well as the outline of this thesis. Because the Ph.D. project was explicitly carried out in an industrial context, the context and problems of integration and testing in current industrial practice were used as main drivers for the research.

Section 1.1 describes the context and the current industrial practice of high-tech multi-disciplinary system development. In Section 1.2, it is shown how several conflicts and trends in the current way of working result in an integration and test problem that is disadvantageous for the most important business drivers: time-to-market and product quality. The influence of integration and testing on these business drivers is explained in more detail in Section 1.3. Section 1.4 identifies three possible solutions for the integration and test problem and shows the resulting effects on the business drivers. The remainder of the thesis focuses on one of these solutions, early integration and system testing, for which several research questions are defined in Section 1.5. Section 1.6 provides an outline of this thesis.

1.1 System development

This section introduces high-tech multi-disciplinary systems and the challenges that manufacturers of such systems need to deal with in their current system development process.

1.1.1 High-tech multi-disciplinary systems

We define a system as a collection of smaller parts, called components, which are ordered and interacting according to a certain architecture. The interaction between the components

results in system behavior, which aims at satisfying requirements, e.g., providing some functionality with a defined performance. High-tech multi-disciplinary systems are systems in which cutting edge technologies from multiple engineering disciplines are integrated in order to meet the strict quality requirements set by the customer. A characteristic of such systems is that they are 'integration intensive', which means that integrating the components and testing the resulting system against its requirements consumes a large portion of the total system development effort. In some cases, the integration and test effort may even exceed the already considerable amount of development effort required to deal with the complexity of the components. Examples of high-tech multi-disciplinary systems are commercial systems such as wafer scanners [ASML 2007], electronic microscopes [FEI Company 2007], and high-speed printers [Océ 2007], medical systems such as magnetic resonance imaging (MRI) scanners [Philips Medical Systems 2007], as well as aerospace systems such as (instruments for) satellites [SRON 2007], airplanes [Airbus 2007], and spacecraft [NASA 2007].

In this Ph.D. project, the wafer scanners from ASML, used worldwide to manufacture integrated circuits (chips) such as processors and memory, are used as carrier examples of high-tech multi-disciplinary systems. Wafer scanners use light to transfer a lithographic image (a pattern corresponding to one layer of a chip) from a mask onto the surface of a silicon wafer with nanometer accuracy. The light passes through an optical system that shrinks the pattern image before it is projected onto the wafer. A wafer scanner is highly multi-disciplinary; it features a complex lithographic process (chemistry, physics) which requires the latest lens technologies (optics), materials with high demands on thermal and dynamic properties (mechanics), fast and accurate motion control (electronics, embedded software), complex metrology models for measurement and calibration (mathematics), and a large amount of software to control the system (computer science).

Figure 1.1 shows a new type of wafer scanner that is currently under development within ASML. This wafer scanner uses extreme ultra violet (EUV) light for exposing wafers, which has a much smaller wavelength than the laser light used in the current wafer scanners, enabling higher accuracies in the exposure process. One of the most important technical challenges in the development of this lithography system is that EUV light is absorbed by nearly all materials, including air. This implies that the refractive optics used in the current wafer scanners need to be replaced by reflective optics and that the lithography process needs to be performed under strict vacuum conditions.

1.1.2 Current system development process

In current industrial practice, the system development process is usually based on a 'divide and conquer' strategy, in which the system is decomposed into smaller parts that are separately developed. This system decomposition may be applied on multiple hierarchy levels, e.g., systems into subsystems, subsystems into modules, and modules into components. In this Ph.D. project, we only consider the system decomposition between two hierarchy levels, in which the higher level is referred to as *the system* and the lower level is referred to as *the components*.

When a system is decomposed into components at the start of the development process, complementary activities are required to obtain the intended system realization at the end of

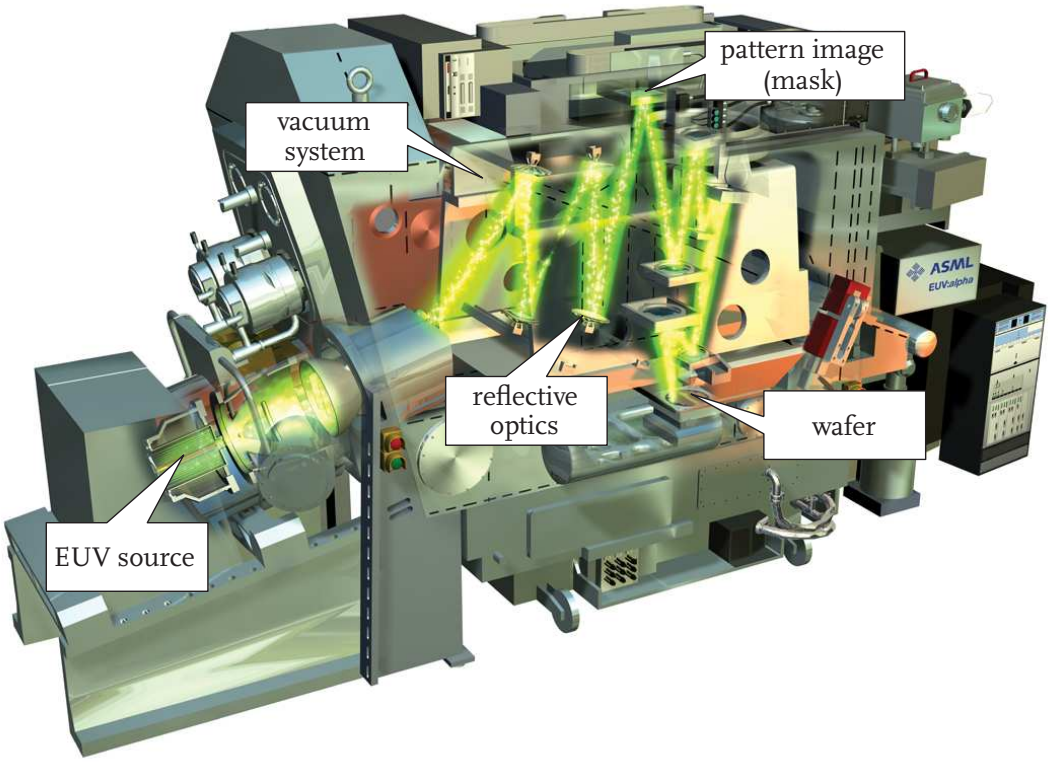


Figure 1.1: ASML EUV wafer scanner

the development process. These complementary activities take place in the *integration and test* (I&T) phases, in which the system is built up by combining the realized or implemented components and, subsequently, tested against the system requirements that were defined initially. This system development process corresponds to the well-known and widely used V-model [Rook 1986] as shown in Figure 1.2. The left-hand side of the V-model corresponds to the decomposition of the system into components, including requirements definition and design phases as depicted by the boxes. The right-hand side of the V-model corresponds to the integration of component realizations resulting in the system realization, with different test phases as counterparts of the phases on the left-hand side.

During the system development process, manufacturers of high-tech multi-disciplinary systems such as ASML are faced with many challenges. We consider the following four challenges that, as explained in the next section, have significant impact on the integration and test phases.

Challenge 1: System complexity

A system consists of many components of different disciplines, interconnected by many interfaces in order to enable the interaction required for the system functionality and performance. The system complexity is too large to be comprehended by a single person, so therefore system level specifications (e.g., throughput) are decomposed into smaller component specifications, which express the contribution of each component to the system level specifications in terms of the corresponding engineering

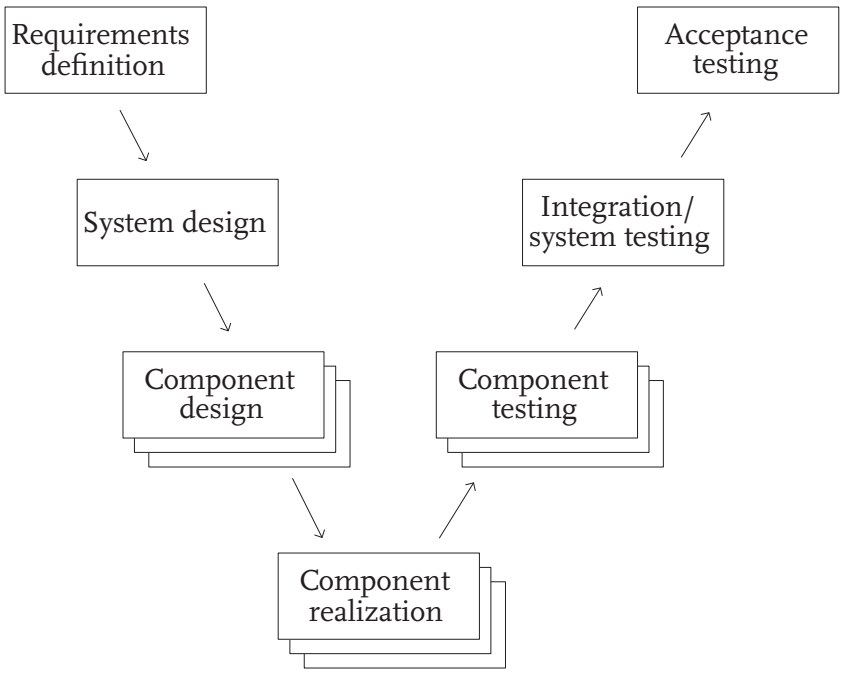


Figure 1.2: V-model

discipline (e.g., acceleration and electrical power). Even the complexity of a component can be significant, especially when a component contributes to multiple system level specifications, or when emerging technologies from particular engineering disciplines are used for the first time. When different persons are responsible for different components, adequate specification and communication skills are required to understand the relations between components and the possible implications of component design changes for other components. Many industrial system development processes, e.g., at ASML, are document-based, i.e., documentation is used to keep track of the requirements and designs of the system and components. To be suitable for understanding the system complexity, the documentation should specify the intended behavior of the components and the integrated system as completely, consistently, and unambiguously as possible. Adequate specification of intended behavior is also crucial for the I&T phases, since the actual behavior (as observed by performing tests) is compared to the intended behavior to determine the outcome of a test, i.e., pass or fail. In current industrial practice, however, the documentation may be incomplete, inconsistent, and ambiguous. As a result, understanding the system complexity and determining test outcomes based on documentation only may become quite difficult.

Challenge 2: System overview

It is difficult to retain a good overview of the system throughout the complete development process. Initially, the global system requirements and system design are reasonably well-known, based on customer requirements and the estimated technological capabilities. However, the complex and multi-disciplinary nature of the system obstructs

specification of complete sets of requirements at each hierarchy level, resulting in less certain predictions of the actual system behavior. Obtaining complete requirements and predicting the actual system behavior is especially difficult when emerging and not yet mature technologies have to be introduced in the system, because of the limited experience with these technologies. As a result, the system overview fades away during the separate component development processes, until it 'suddenly emerges' when the components are integrated and the actual system behavior becomes visible, regardless of the question whether or not this actual behavior corresponds to the intended system behavior. Without system overview, it is difficult to ensure that the system being developed will eventually show the intended behavior and that it will satisfy the requirements. Empirical studies such as [Curtis et al. 1988; Herbsleb and Kuwana 1993] also observed that engineers have difficulties in obtaining and retaining system overview throughout the development process, and that 'project gurus' with good system overview are scarce but essential project resources.

Challenge 3: Resource limitations

The development process of a high-tech multi-disciplinary system requires large amounts of time and money in terms of human resources, material costs, and other R&D costs. However, these resources are limited by the amount of available man hours and by the total R&D budget that can be spent. Therefore, it is preferred that the resources are used wisely and not in vain. This means that the amount of rework in the system development process (e.g., changes in requirements, redesigns, multiple versions of realizations) should be minimized, aiming at building the system 'first time right'. Herein lies a paradox: resource limitations dictate that a system should be built 'first time right', which often requires a more complete system overview than one can obtain within the resource limitations.

Challenge 4: Time-to-market

The business drivers of a company are often influenced by the market conditions in which the company is operating. Business drivers can be characterized in terms of time-to-market (T), product quality (Q), and costs (C), ordered according to their relative importance for the company [De Jong et al. 2007a]. For instance, in the aerospace industry, product quality is essential and time-to-market seems less important, resulting in business drivers that can be characterized by Q-C-T, meaning that quality is most important, then costs, and time-to-market is least important (but not totally unimportant). For manufacturers of many commercial high-tech multi-disciplinary systems, however, the market conditions often require that production-ready systems are delivered to the customers at the earliest possible date. At least in the semiconductor equipment industry in which ASML is operating, a short time-to-market is of utmost importance for commercial success. As a result, the business drivers of lithographic equipment manufacturers such as ASML can be characterized by T-Q-C. In practice, the importance of time-to-market results in strict constraints on the system development process, such as agreements between the manufacturer of the system and the customer to ship the system before a fixed deadline, with high financial penalties for exceeding this deadline.

1.2 Integration and test problem

By looking at how the challenges mentioned in the previous section are dealt with in the current system development process, several conflicts between the challenges can be identified. These conflicts are disadvantageous for the I&T phases, e.g., resulting in unnecessary high costs and long lead times for integration and testing as well as delayed system shipments, i.e., increased time-to-market.

Conflict 1: System decomposition increases system complexity and reduces system overview
System decomposition results in smaller parts that are easier to comprehend by a single person. Although this is necessary to deal with system complexity, it may as well increase system complexity since more components are developed and, accordingly, more requirements and design documents are created. As a result, the relations between components, e.g., physical relations and interfaces, and the corresponding relations between documents become less clear and more difficult to understand and manage, resulting in a reduced system overview. Sometimes, the exact relations between the components remain unclear until the realized components are integrated and tested, i.e., the system overview 'suddenly emerges' when the actual system behavior becomes visible. Moreover, integration problems that were not foreseen during development might show up only when the system is integrated and tested.

Conflict 2: Limited resources and time-to-market pressure hinder complete system overview
Due to system complexity, a large amount of time would be required to obtain a complete set of system requirements (i.e., complete system overview) before starting the design and realization phases of the system development process. Besides that achieving complete system overview is nearly impossible in most cases, resource limitations and time-to-market pressure further reduce the time available for obtaining system overview. In practice, the developers may deliberately start designing even if some requirements are not yet known. During the design, realization, and integration phases, their understanding and overview of the system improves, which helps to specify the requirements that are still missing. Although this approach might not result in building the system 'first time right', it might be a necessary step to get to a complete system at all. In some cases, also the I&T phases may already start before all requirements are known, and, moreover, some specifications might not become clear until some tests are executed on the system that is integrated. One can observe a paradox in the fact that tests are performed even when not all requirements are known, while, by definition, testing should check whether the requirements are met by the system. In practice, this paradox is resolved by engineers who, based on previous experiences, analogies, and engineering intuition, suggest tests even for parts of the system that are not yet sufficiently covered by the requirements. Practice shows that some of these tests appear to be quite successful for the validation or falsification of the desired system behavior and for clarifying some of the requirements that are still missing. Other tests, however, yield little information and, as such, can be characterized as unfortunate waste of resources.

Conflict 3: Insufficient system overview leads to rework and increased time-to-market

Although tests help to increase system overview, tests that do not yield the expected results or tests that detect unforeseen or unsolved problems in the system usually result in additional rework in the system development process. This rework may be limited to retesting some functionality, but it can also involve time consuming and costly rework in the requirements, design, and realization phases. Besides the costs of rework in terms of time and money, it also increases the pressure on the I&T phases of the involved components. In this way, the I&T phases get 'squeezed' between, on the one hand, the component development phases that suffer from time consuming rework and corresponding late component deliveries, and, on the other hand, the time-to-market pressure represented by the fixed system shipment date agreed with the customer. Furthermore, the I&T phases remain on the critical path of the system development process, meaning that any delay in the I&T phases directly influences the time-to-market and threatens timely shipment of the system. As a result, the quality of the shipped systems is sufficient for operation, but additional testing after the shipment date is required to optimize the quality. Figure 1.3, taken from the TANGRAM project plan [Brugman and Beenker 2003], shows how the I&T phases are squeezed between component development and system shipment, and how testing continues after shipment to improve the quality of the shipped system.

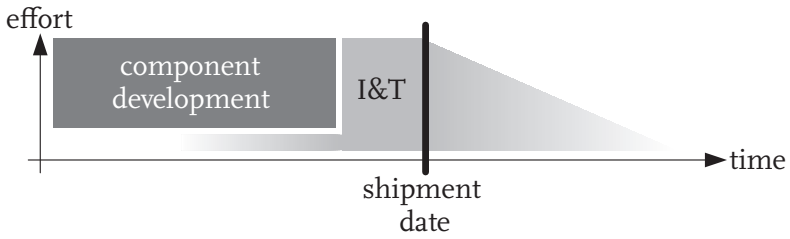


Figure 1.3: I&T phases 'squeezed' between component development and system shipment

As the system development challenges described in Section 1.1 grow, these three conflicts for the I&T phases also become more critical. As a result, the role of the I&T phases within the system development process becomes crucial for successful delivery of systems with sufficient quality within the resource and time-to-market constraints.

These trends related to integration and testing are also recognized in several studies found in literature. For example, [Muller 2007] provides a good overview of the complexity of the I&T process and the many relations it has with system development and other organizational aspects that are also becoming more and more complex. [Bratthall et al. 2000] shows that the main effort of the industrial system development process is shifting from the design and implementation phases to the I&T phases. Problems detected during the I&T phases can only be fixed late in the development process, or even worse, at the customer's site, which can be up to 100 times more expensive than detecting and fixing the problems during the requirements and design phases [Boehm and Basili 2001]. Company losses related to system failures may exceed 10% of the company's turnover [Sorqvist 1998]. Together, the costs for

testing and the rework required to fix the detected problems take up a significant portion (over 50%) of the total system development costs [Boehm and Basili 2001; Engel et al. 2004].

The overall conclusion related to these conflicts and trends is that the current system development process suffers from an I&T problem, in which the I&T phases have a large and growing disadvantageous influence on the T-Q-C business drivers. As described in [Prins 2004], research is required to solve this I&T process, which immediately leads to the main objective of this thesis.

> The main objective of this thesis is to show how the disadvantageous influence of the I&T phases on the time-to-market, product quality, and costs (T-Q-C) for developing high-tech multi-disciplinary systems can be reduced.

In order to reach this objective, the influence of the I&T phases on the T-Q-C business drivers needs to be investigated in more detail and possible solutions to the I&T problem need to be identified, which is done in the next sections of this chapter.

1.3 Business drivers

By relating Figure 1.3 to the T-Q-C business drivers, the disadvantageous influence of the I&T phases on these business drivers and the effects of possible solutions can be visualized. In this intuitive explanation, we focus on the two most important business drivers for lithographic equipment manufacturers such as ASML, time-to-market T and product quality Q. Initially, the costs C are not taken into account. For time-to-market T, the relation to Figure 1.3 is simple: the time-to-market T is the time between the start of component development and the system shipment date. For product quality Q, the relation to Figure 1.3 is less obvious, and another interpretation of the figure is needed.

Research on integration and test strategies often uses *remaining risk* as a measure for product quality Q [De Jong et al. 2006] and as a means to determine when to stop testing [Williams and Ambler 2002; Boumen et al. 2006]. As explained in [De Jong et al. 2006], the remaining risk in a system, denoted by R, is defined as the sum of the risks for all possible faults in the system. The risk for a certain fault is defined as the probability that the fault is present in the system multiplied by the impact of the fault when it is present and when it manifests itself in the system behavior, e.g., the costs of a system failure. Possible faults, and thus the risk related to them, are *introduced* in the system via component development (i.e., faults in a component) and via component integration (i.e., faults in the interaction between components). By testing a component or a system of integrated components and by fixing the detected problems, the risk in the components and their interaction, and thus the remaining system risk R, is *reduced* (i.e., the quality Q is improved). Before system risk can be reduced by testing, however, it has to be *revealed* in the system, i.e., possible faults, whose introduction may remain unnoticed during component development and integration, have to manifest themselves in the tested system behavior to be detected. In many cases, some time elapses between the moment of introducing risk and the moment that this introduced risk reveals itself, e.g., an interface error that is overlooked during component design may only become visible when the component realization is integrated into the system. The way that system

risk R is introduced, revealed, and reduced during different phases of system development can be visualized in a so-called risk profile [De Jong et al. 2006], which graphically represents the remaining system risk over time.

Figure 1.4 is another interpretation of Figure 1.3 and shows typical risk profiles for each phase of the current system development process. Here, we only consider risk on the system level, caused by faults in components that influence the system behavior and by faults in the component interaction. This means that we do not consider component risk that does not influence system behavior, assuming that this 'internal' component risk is dealt with during component development and component testing. The figure shows when the system risk R, introduced by component development and component integration, reveals itself and how it is reduced by testing on the system level and by fixing the detected problems. For simplicity reasons, linear abstractions of the risk profiles are used, which is sufficient for this intuitive explanation of the influence of the I&T phases on the T-Q-C business drivers. Note that the system risk R on the y-axis considers both revealed risk (for risk profiles above the 'zero risk' line, denoted by R_o) and reduced risk (for risk profiles below R_o).

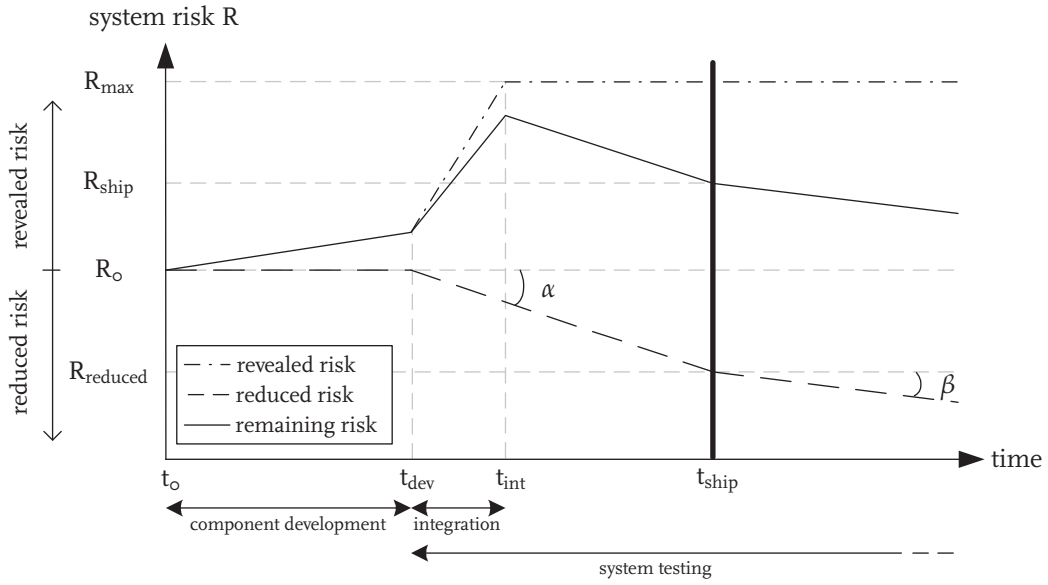


Figure 1.4: Typical risk profiles for current system development

The dash-dotted line in Figure 1.4 depicts the risk revealing profile for the development and integration phases. During the component development phase from t_o to t_{dev}, the focus is typically more related to the component level rather than to the system level, and only a small portion of the total system risk is revealed. The largest portion of the total risk on the system level, however, is revealed when the developed components are combined and the actual system behavior becomes visible, which happens during the integration phase from t_{dev} to t_{int}. Since only system level risk is considered here, the figure shows a risk revealing profile that slowly increases between t_o and t_{dev} (component development phase), after which it increases significantly between t_{dev} and t_{int} (integration phase). When all components are integrated, we assume that all potential system risk is revealed, i.e., it can be reduced by test-

ing and by fixing the detected problems. In the figure, this is depicted by the risk revealing profile that remains constant at its maximum R_{max} after t_{int}.

In contrast to the risk revealing profile described above, the dashed line in Figure 1.4 depicts the risk reduction profile for the test phase. It is important to note that the risk reduction profile depends on the risk revealing profile in the following way. In this intuitive explanation, we assume that system risk can only be reduced by performing tests on a combination of at least two realized and integrated components and by fixing the detected problems. This means that the risk reduction profile for the test phase depends on the point at which the integration phase starts, t_{dev} in the figure. Before t_{dev}, no component realizations are integrated and no system risk can be reduced by testing, resulting in a risk reduction profile that remains at R_o. At t_{dev}, the first components are integrated and system tests can be performed to reduce the system risk, represented in the figure by the negative values of the dashed line. Testing continues after all components are integrated, i.e., after t_{int}. How much and how fast risk can be reduced by testing depends on the quality of the system tests, the efficiency of their execution, and the time needed for diagnosing and fixing the detected problems. This rate of risk reduction over time is represented in the figure by the angle α of the linear abstraction of the risk reduction profile. Depending on the risk reduction rate, a certain amount of risk $R_{reduced}$ can be reduced before the (fixed) system shipment date t_{ship}, depicted by the bold vertical line similar to Figure 1.3. After t_{ship}, when the system is operating at the customer site, the risk reduction rate often decreases because less test time is available and the conditions for testing are less optimal when compared to the test phase before system shipment. This lower risk reduction rate for testing after shipment is represented in the figure by the angle β, which is smaller than α for testing before shipment. Note that we assumed here that only one system was used for testing. Practical experience shows that when multiple systems of the same type are shipped and used concurrently at different customer sites, e.g., during beta testing, more problems are detected and the total risk reduction rate for that particular system type may even increase after system shipment.

The overall remaining risk profile for the system development process is obtained by combining the risk revealing profile and the risk reduction profile, which results in the solid line in Figure 1.4. Between t_o and t_{dev}, the remaining risk is equal to the revealed risk during component development. Between t_{dev} and t_{int}, a significant amount of risk is revealed during integration while some risk is reduced by testing, resulting in an overall remaining risk that still increases. After t_{int}, the remaining risk profile follows the risk reduction profile with risk reduction rates α and β. The remaining risk at system shipment date t_{ship} is denoted by R_{ship}, which is used as a measure for the product quality Q: a lower R_{ship} implies a higher quality Q of the shipped system.

1.4 Solutions to the I&T problem

Using Figure 1.4, possible solutions can be identified to reduce the disadvantageous influence of the I&T phases on the T-Q-C business drivers. The following subsections describe three possible solutions, for which the resulting effects are expressed and visualized in terms of time-to-market T and remaining risk R, similar to Figure 1.4.

1.4.1 Development phase improvement

This solution focuses on the development phase, aiming at better requirements, designs, and realizations of the components. With better component designs and realizations, less risk is introduced (and thus revealed) during component development and component integration, meaning that less risk needs to be reduced by testing. Under the assumption that the quality and efficiency of the test phase and thus the risk reduction rates do not change, Figure 1.5 shows the changes in the risk profiles compared to those in Figure 1.4. For easier comparison, some parts of Figure 1.4 are included in Figure 1.5, namely the risk revealing profile (grey dash-dotted line), the remaining risk profile (grey solid line), and the original shipment date t_{ship} (grey and bold vertical line).

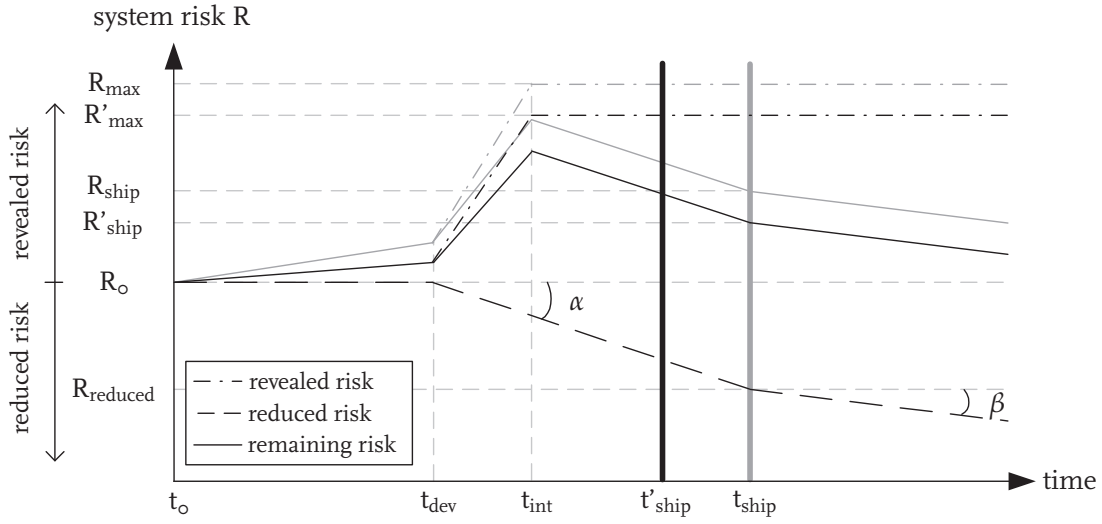


Figure 1.5: Risk profiles for development phase improvement

Compared to Figure 1.4, the total amount of revealed risk decreases from R_{max} to R'_{max}, and therefore the remaining risk profile (black solid line) is shifted downwards as well. Looking at the remaining system risk at time of shipment for the current way of working, R_{ship} at time t_{ship} in Figures 1.4 and 1.5, the same product quality level is now reached at time t'_{ship} instead of t_{ship}, meaning a shorter time-to-market T. If the shipment date remains at t_{ship}, however, the system can be shipped with less remaining risk, R'_{ship} instead of R_{ship}, meaning an improvement of quality Q. In this way, this solution enables a choice of how the T-Q-C business drivers are influenced, resulting in an earlier shipment date between t'_{ship} and t_{ship} with corresponding risk between R'_{ship} and R_{ship}.

In current industrial practice, improvements to the development phase are usually translated into allocating more resources for component development. However, adding more resources is not always possible (see Challenge 3 in Section 1.1) and might even have negative effects [Brooks 1995]. Besides adding resources, the component development process itself can also be improved. For instance, techniques can be applied from areas such as systems engineering [Martin 1996; INCOSE 2006], requirements engineering [Hull et al. 2005], model-based design [Gomaa 2000; The Mathworks – SIMULINK 2007], and code

generation [De Wulf et al. 2005; Huang et al. 2006], which already receive major attention in research. Although these approaches may show promising research results, large scale implementations in industrial practice are scarce and, at least at ASML, are not expected in the near future, meaning that the I&T problem still remains. Furthermore, it is unknown whether these approaches can deal with practical challenges such as described in Section 1.1, and whether they are sufficiently effective at preventing integration problems to solve the I&T problem. Moreover, one explicit constraint of the TANGRAM project was that the current way of working for the requirements, design, and realization phases (i.e., the left-hand side of the V-model) should be taken for granted and may not be changed [Brugman and Beenker 2003]. Because of these reasons, this solution is not considered in this Ph.D. project.

1.4.2 Test phase improvement

A second solution lies in the test phase of the system development process. By improving the efficiency of the test phase activities, risk could be reduced faster, i.e., the risk reduction rate increases. Under the assumption that the risk revealing profile (dash-dotted line) does not change, Figure 1.6 shows the changes in the risk profiles compared to those of Figure 1.4. Again, some parts of Figure 1.4 are included (in grey) for easier comparison.

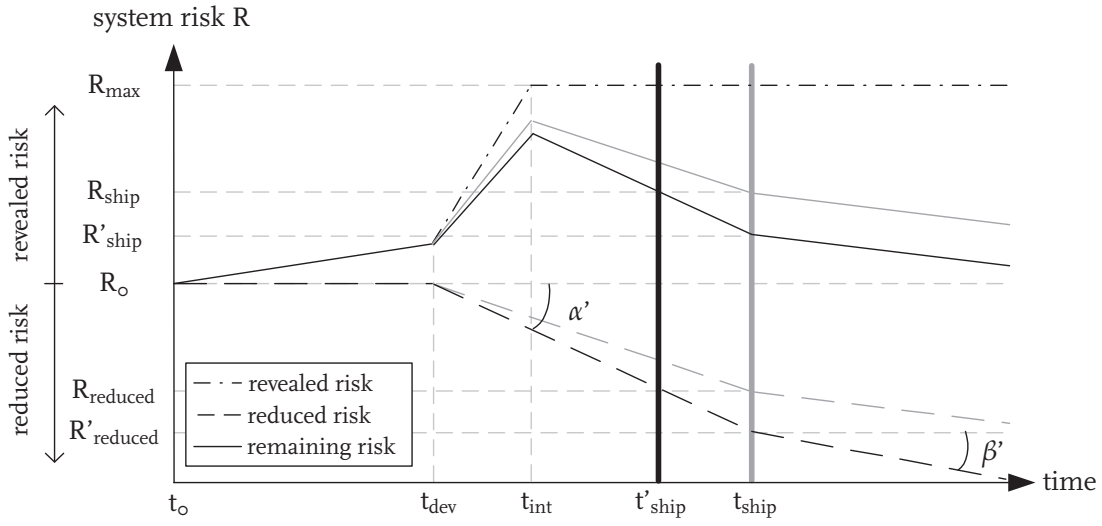


Figure 1.6: Risk profiles for test phase improvement

Compared to Figure 1.4, the angles representing the risk reduction rates before and after system shipment, increased to α' and β', respectively. As a result of the increased risk reduction rate, more risk can be reduced within the same test time and $R_{reduced}$ increases to $R'_{reduced}$. The remaining risk profile (black solid line) shows the improvements to T and Q, i.e., earlier shipment between t'_{ship} and t_{ship} with corresponding risk between R'_{ship} and R_{ship}.

One way to increase the risk reduction rate is to allocate more resources for testing, diagnosis, and problem fixing, however also these resources are limited, e.g., a limited number of test benches and prototypes on which tests can be performed. Another way to achieve this is

to improve the quality of the tests and the efficiency of test execution, diagnosis, and problem fixing, e.g., by automating test and diagnosis activities, by using a flexible test environment allowing easy configuration of (exceptional) test conditions, or by optimizing the sequence of all activities in the I&T process. These improvements were investigated in other parts of the TANGRAM project, see [Tretmans 2007] for an overview.

Although this solution may enable faster risk reduction, the point at which the test phase can start remains at t_{dev}, since the tests still require that at least some components are realized and integrated. When the starting point for system testing remains at t_{dev}, it is questionable whether the improvements to the risk reduction profile after t_{dev} provide enough compensation for the large amount of risk that is only revealed in the component integration phase, which also starts after t_{dev}.

1.4.3 Early integration and testing

The point at which the test phase can start, t_{dev} in the current way of working, is the focus of the third solution, which aims at *earlier* integration and testing of system risk. When system tests could be performed before the components are realized and integrated, i.e., before t_{dev}, the test effort and corresponding risk reduction could be distributed over a wider time frame. Performing system tests before the system is available requires a method that enables system risk to be revealed earlier and in another way than in current component development and integration, enabling earlier risk reduction by testing. Such a method should provide techniques to represent and integrate the components and the corresponding risks at an early stage, such that system tests can be applied on this early representation of the integrated system. Under the assumption that such a method for *early integration and testing* exists and that the amount of introduced risk and the risk reduction rates do not change, Figure 1.7 shows the changes in the risk profiles compared to those in Figure 1.4. Again, some parts of Figure 1.4 are included (in grey) for easier comparison.

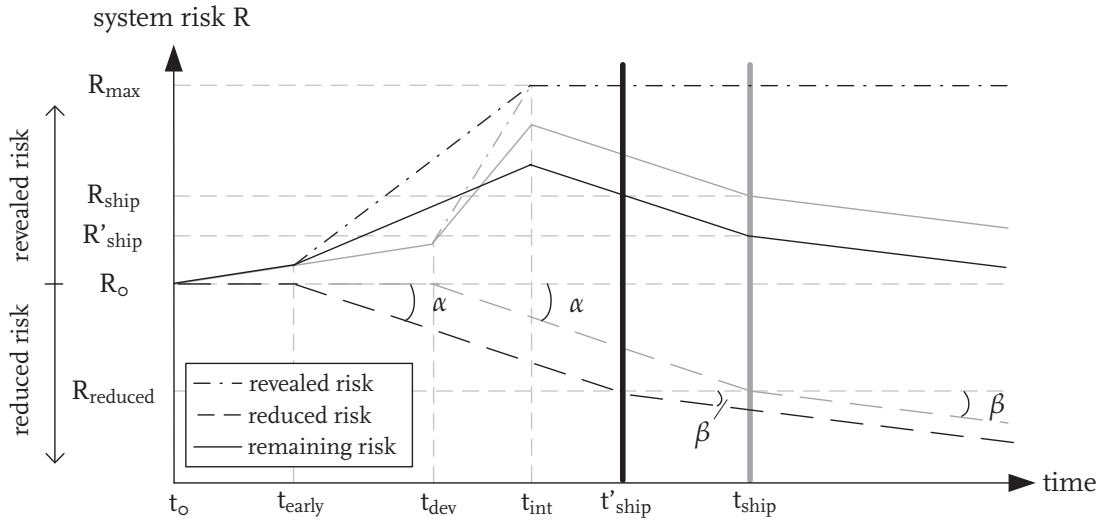


Figure 1.7: Risk profiles for early integration and testing

In Figure 1.4, component integration and system testing could only start at t_{dev}, since the current way of working requires at least some components to be realized and integrated. The assumed method uses other techniques than component realizations to represent, integrate, and test system risk at an early stage, such that these activities can already start at t_{early}. Although the risk revealing profile (black dash-dotted line) starts its major increase earlier, the total amount of revealed system risk R_{max} still depends on the realized and integrated components. Since the development and integration process for component realizations may not be changed, R_{max} and the point in time at which all potential system risk is revealed, t_{int}, are the same as in the current way of working. The system risk that is revealed earlier by means of the early I&T method is immediately available for testing. This means that the test phase and the corresponding risk reduction can start earlier as well, i.e., at t_{early} instead of t_{dev}. This is depicted by the risk reduction profile (black dashed line), which has the same risk reduction rates α and β as in the current way of working. Again, this enables improvements to T and Q by earlier shipment between t'_{ship} and t_{ship} with corresponding risk between R'_{ship} and R_{ship}, as shown by the remaining risk profile (black solid line) in Figure 1.7.

The main challenge of this solution lies in the assumed method that enables early integration and testing of system risk before the actual components are realized and integrated. This thesis takes on the challenge, and investigates the possibilities, requirements, practical applicability and profitability of a method that enables early integration and system testing.

1.5 Research questions

As previously mentioned, the main objective of this thesis is to show how the disadvantageous influence of the I&T phases on the time-to-market, product quality, and costs (T-Q-C) for developing high-tech multi-disciplinary systems can be reduced. In the previous section, three possible solutions to the I&T problem were identified. Development phase improvement is not further investigated, because it already receives major attention in research and because of the explicit constraint of the TANGRAM project that the current way of working for the requirements, design, and realization phases (left-hand side of V-model) should be taken for granted. While test phase improvements are investigated in other parts of the TANGRAM project, this Ph.D. project focuses on the third solution: a method that enables early integration and system testing. This section defines the research questions that need to be addressed when such an early I&T method is proposed, starting with questions related to the main goal of early integration and testing, as discussed in the previous section.

QUESTION 1.1 How can system risk be revealed and subsequently reduced when no component realizations are available?

QUESTION 1.2 How can system risk be revealed and subsequently reduced when only some component realizations are available?

As in all projects organized by the Embedded Systems Institute (ESI) [Embedded Systems Institute 2007], one particular goal of the TANGRAM project is to provide a proof of

concept showing that the proposed methods, techniques, and tools are applicable and profitable in industrial practice. In this so-called 'industry as laboratory' research concept [Potts 1993; Embedded Systems Institute 2006], research is performed in a close relation with the actual system development activities of a carrying industrial partner, which is ASML for the TANGRAM project. This also means that the proposed methods, techniques, and tools should be able to deal with the (not always optimal) conditions and constraints of current industrial practice, a requirement that is not often considered in academic research projects. For example, the fact that the current system development process is based on documents that may be scattered, incomplete, outdated, and may contain information that is ambiguous and inconsistent, should just be accepted and dealt with in some way. Due to this practical context of the TANGRAM project, the answers to the research questions should be supported by a proof of concept showing their industrial applicability and profitability. Assuming that the answers to QUESTIONS 1.1 and 1.2 result in a method for early integration and testing, then the following additional question should be answered as well.

QUESTION 1.3 Is it feasible and profitable to apply the proposed early I&T method in current industrial practice?

The proposed early I&T method adds new integration and test activities to the I&T process in order to reduce time-to-market T or to increase the product quality Q. However, a method that reveals system risk early such that it can be reduced by testing will probably require additional resources during the system development process. For example, a certain amount of time will be needed to represent and integrate the components and the corresponding risks, and to perform tests on this early representation of the integrated system. Since resources are limited, it is important to determine whether application of the early I&T method is profitable and whether the resources required for it are used wisely and not in vain. To determine this profitability, the third and, according to the T-Q-C order, least important business driver, costs C, has to be taken into account, which was not the case in Section 1.3. By quantifying the costs C required to enable the benefits of shorter time-to-market T and higher product quality Q, the profitability of early integration and testing can be determined. This supports the decision making process of where and when the I&T process can profit from early integration and testing, as expressed in the second set of research questions for this thesis.

QUESTION 2.1 Which activities in the current I&T process can be supported by the early I&T method?

QUESTION 2.2 When is it profitable to apply the early I&T method?

QUESTION 2.3 Is it feasible in current industrial practice to quantify the costs of the early I&T method and to decide where and when the method should be applied?

In the remainder of this thesis, QUESTIONS 1.1 and 1.2 are answered by proposing methods, techniques, and tools to enable early integration and testing. QUESTIONS 2.1 and 2.2 are

answered by proposing methods, techniques, and tools to quantify the costs and profitability of applying the early I&T method. To provide an answer to QUESTIONS 1.3 and 2.3, the proposed methods, techniques, and tools were applied to a relevant industrial case study, which is used throughout the thesis as an example. This case study involves the EUV wafer scanner as shown in Figure 1.1.

1.6 Outline

Due to the practical context and constraints of this Ph.D. project, it is a research project with a strong engineering component. The engineering nature of the Ph.D. project is also reflected in this thesis, in which theories and techniques from specific research domains are used and combined to answer the research questions. The proposed early I&T method is based on theories and techniques from research domains related to *model-based engineering*, in which computer models are used to describe, analyze, and test components and systems.

Chapter 2 first describes the current system development process in more detail. Subsequently, theories and techniques from several research domains related to model-based engineering are used and combined to answer QUESTIONS 1.1 and 1.2, resulting in the *model-based integration and testing method*. Finally, this method is instantiated with a particular paradigm, corresponding mathematics, and tools, which are used in the subsequent chapters to answer QUESTION 1.3 on the applicability and profitability of the method in industrial practice.

These subsequent chapters discuss the three main activities of the method in more detail: system analysis (Chapter 3), component testing (Chapter 4), and integration and system testing (Chapter 5). After some rationale and background for the discussed activity, each of these three chapters describes theory and techniques from related research domains that are used in the method. Subsequently, each chapter describes the practical side of applying the discussed activity to the EUV case study, providing an answer to QUESTION 1.3.

Chapter 6 focusses on QUESTIONS 2.1 through 2.3. The chapter starts with a description of where, i.e., in which activities, the current I&T process can be improved by applying the early I&T method described in the previous chapters, answering QUESTION 2.1. Then it is shown how integration and test sequencing techniques can be used to quantify the costs of various I&T processes, in order to decide when it is profitable to apply early integration and testing in the I&T process, answering QUESTION 2.2. Subsequently, QUESTION 2.3 is answered by applying this quantitative decision making process to the I&T process of a new version of the EUV wafer scanner.

Finally, some concluding remarks are given in Chapter 7.

CHAPTER 2

Model-based integration and testing

The 'how' and the 'with what'...

As described in the previous chapter, this Ph.D. project focuses on a method for early integration and testing. Before answering the research questions related to this method, a description of the current system development process is given in Section 2.1, specifically focusing on the I&T activities involved in the process.

Section 2.2 introduces the *model-based engineering* approach and the related research domains that provide several model-based theories and techniques. This model-based engineering approach is used as the 'enabler' for early integration and testing, resulting in several model-based I&T techniques proposed in Section 2.3 as answers to QUESTIONS 1.1 and 1.2. By combining the proposed model-based I&T techniques with the current system development process, the *model-based integration and testing method* or MBI&T method is obtained, which enables earlier and faster integration and testing with lower costs.

Before the MBI&T method can be applied to real industrial I&T problems, it needs to be instantiated with a paradigm and related mathematical techniques and tools which are suitable to perform the MBI&T activities. Section 2.4 describes the instantiation of the MBI&T method. In the same section, we present our hypothesis that a particular instantiation of the MBI&T method, based on the paradigm of concurrent processes and process algebra techniques and tools, is suitable to perform all MBI&T activities in industrial practice regarding component interaction and time behavior. In the remainder of this thesis, this hypothesis is tested by applying the particular instantiation of the MBI&T method to the EUV case study, thus answering QUESTION 1.3.

2.1 Current system development process

This section describes the current system development process, which is based on the V-model as shown in Figure 1.2. Although the focus of this figure was on the different *phases* of system development, this section concentrates on the different *representations* of components and systems. Three different representations are considered: requirements, designs, and realizations. Requirements, denoted with *R*, typically specify the required functionality

and performance, as well as other constraints that should be satisfied by a component or system. Requirements usually originate either from customer demands or from requirements and designs on a higher hierarchy level. During the requirements phase, customer demands are translated into technical requirements taking the estimated technological capabilities into account, and the influence of higher level requirements and design decisions on the lower level requirements is determined. For example, the required throughput of a system is translated into required accelerations and thus into required electrical power of the motors, and the architecture of the system design dictates the interfaces that each component should provide. Designs, denoted with D, typically specify how the component or system should be built in order to satisfy its requirements, e.g., the intended implementation of the provided interfaces and the internal component behavior. Higher level designs often also contain the decomposition into smaller parts and the corresponding translation of higher level requirements and design decisions into requirements for the smaller parts, as well as the architecture, layout, and intended interaction of the smaller parts. Realizations, denoted with Z, are the real components (e.g., mechanics, electronics, software) and systems consisting of integrated components, which are built according to the design and should satisfy the requirements defined for it.

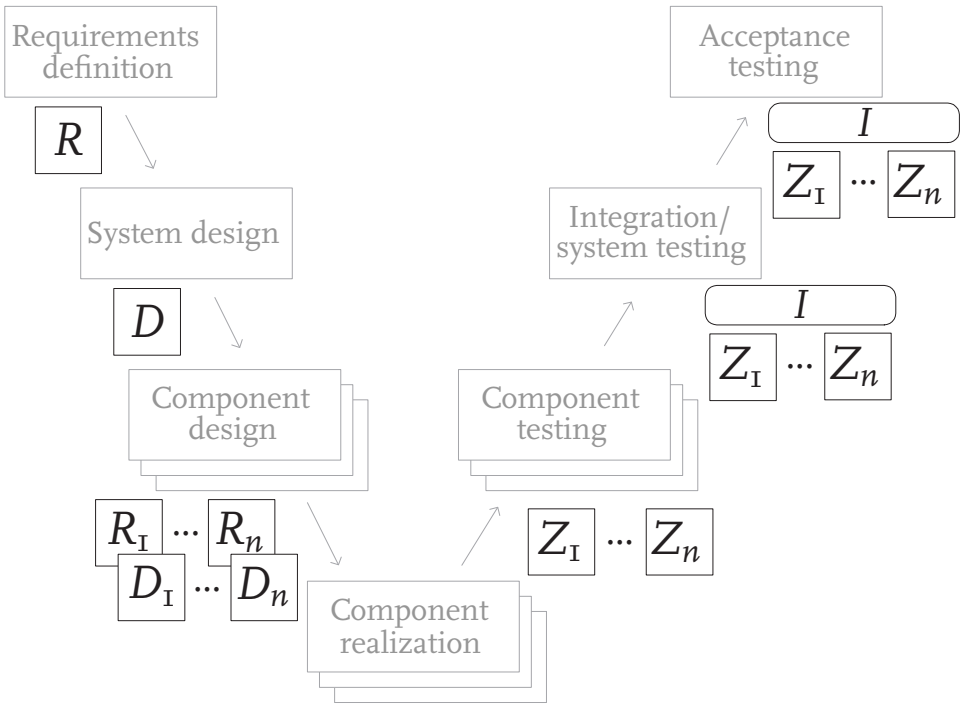


Figure 2.1: V-model: from phases (in grey) to representations (in black)

Figure 2.1 shows how these representations (in black) relate to the phases of the V-model from Figure 1.2 (in grey). The figure considers the development of a system S that consists of n components $C_{1..n}$ (in this thesis, a set $\{X_1, \ldots, X_n\}$ is denoted by $X_{1..n}$). The initial phases of system development result in the system requirements R and the system design D, which form an early representation of system S. Subsequently, each component $C_i \in C_{1..n}$ is

separately developed, resulting in three different representations of the component, namely the requirements R_i, the design D_i, and the realization Z_i. On the right-hand side of the V-model, each component realization Z_i is first tested in isolation in the component testing phase. Subsequently, the component realizations $Z_{1..n}$ are integrated such that the integrated system can be tested to determine whether it corresponds to system design D and whether it satisfies the system requirements R and finally the customer demands.

The integration of component realizations $Z_{1..n}$ is achieved by using a certain infrastructure I, graphically represented by a rounded rectangle in Figure 2.1. Anything that is needed to connect components is considered as infrastructure, e.g., nuts and bolts (mechanical infrastructure), signal cables (electronic infrastructure), or communication networks (software infrastructure). By connecting the individual components, the infrastructure I establishes the component interaction according to the system design D in order to fulfill the system requirements R. In this thesis, different types of infrastructure will be introduced. For now, it is sufficient to abstract from these different types of infrastructure and consider only the 'generalized' infrastructure I. The integration of realizations $Z_{1..n}$ by means of infrastructure I is denoted by $\{Z_{1..n}\}_I$. The reason why the infrastructure is not considered as a component C_i is explained later in this chapter.

A graphical representation of the current development process of system S is shown in Figure 2.2, corresponding to a 'flattened' version of the V-model. The arrows depict the different development phases and the boxes depict the different representations of systems and components. The rounded rectangle depicts the infrastructure I that connects the component realizations $Z_{1..n}$. For simplicity, the figure shows a 'sequential' development process per component, i.e., a phase only starts when the previous phase has been finished. In practice, however, different phases of the development process are executed in parallel, e.g., the design phase may already start while not all requirements are completely defined and understood. As previously mentioned, this is often necessary due to the time-to-market pressure to deliver a system with sufficient quality in time. Furthermore, the real system development process has a more incremental and iterative nature, involving multiple versions of the requirements, designs, and realizations, and feedback loops from certain phases to earlier phases, e.g., from the realization phase back to the design phase.

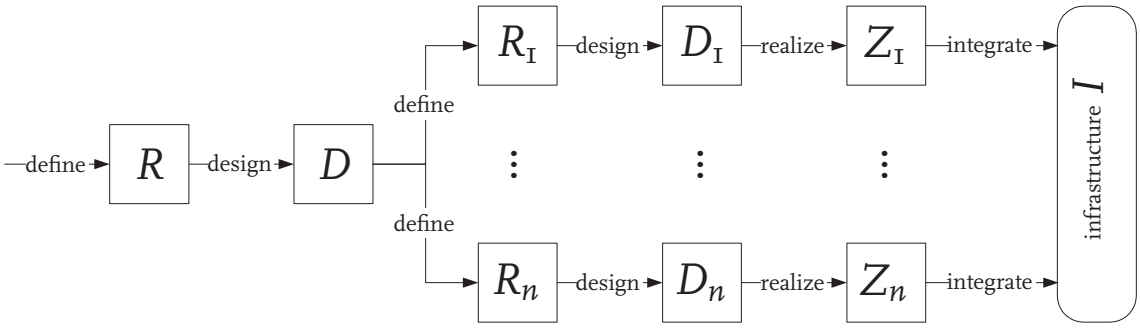


Figure 2.2: Current system development process

As mentioned in Chapter 1, this thesis and therefore also Figure 2.2 only considers the system decomposition between two subsequent hierarchy levels, referred to as *the system* and *the components*. Although real systems usually have more than two hierarchy levels, most system development activities focus either on just one hierarchy level (e.g., system requirements definition and component development) or on the relation between two subsequent hierarchy levels (e.g., system design and system integration). This means that Figure 2.2 can be applied to all hierarchy levels of the real system by mapping the considered hierarchy levels onto the generic system and components hierarchy levels in the figure.

In the remainder of this section, we focus on the possible I&T activities in the current system development process as shown in Figure 2.2. In particular, we only consider I&T activities that aim at detecting problems at the system level, i.e., reducing system risk, as explained in Section 1.3. This means that we do not consider activities related to component development and testing, assuming that the 'internal' component risk is dealt with during the component development and testing phase. In the current system development process, two I&T activities on the system level can be identified, namely requirements and design analysis and system testing, which are described in the following subsections.

2.1.1 Requirements and design analysis

In the current system development process, requirements and design analysis can be applied to check the consistency between requirements and designs on the component level and those on the system level. This consistency heavily depends on the decomposition from system design D into the component requirements $R_{1..n}$. In order to determine this consistency, the separate component requirements $R_{1..n}$ have to be combined (i.e., integrated) and interpreted together. The same holds for the separate component designs $D_{1..n}$. The combined interpretation of $R_{1..n}$ or $D_{1..n}$ can then be be compared (i.e., tested) to the requirements R and design D on the system level, which is graphically represented by the dashed arrows in Figure 2.3, in which the not involved component representations are greyed out. An example of requirements analysis involves understanding the relation between system level requirements R and component requirements $R_{1..n}$ in both directions, which is called requirements traceability [Hull et al. 2005]. An example of design analysis is that when the system design D specifies a certain architecture with corresponding interfaces between the components, the component designs $D_{1..n}$ should comply with this architecture and these interfaces.

Since most of the requirements and designs are currently captured in documents, these activities usually boil down to reviewing and comparing lots of documents, which is a tedious and difficult task. The integration of documents is a rather abstract form of integration, which requires more activities than just combining the components in order to determine the emerging system behavior. For example, the result of integrating two component design documents is just a larger document which does not immediately show the integrated system behavior. Instead, the engineers should create a 'mental' model to interpret the behavior of the integrated components and to check its consistency with the intended system behavior, which requires significant knowledge of the components as well as system overview. In current industrial practice, automated tooling for document-based requirements and design

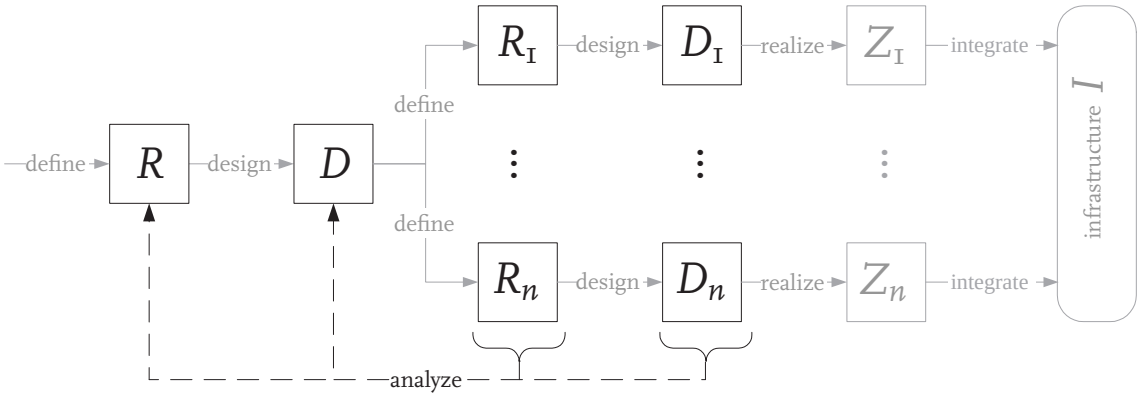


Figure 2.3: Requirements and design analysis

analysis is available, e.g., [Requirements Assistant 2007], but scarce, so these activities are usually performed by reviewing and discussing each document and by applying assessment techniques such as failure mode and effect analysis (FMEA) to identify risks in the system design. Practical experience shows that the available time for reviewing and the system overview are often insufficient to detect and fix all problems regarding the relations between higher level and lower level documents. This means that some problems remain hidden in the design and may only be discovered during testing, or even worse, during operation at the customer site.

2.1.2 Integration and system testing

While the integration of requirements or design documents is rather abstract, the integration of component realizations $Z_{1..n}$ results in the real system, i.e., $\{Z_{1..n}\}_I$. When this integrated system realization is available, it can be tested against the system requirements R and the intended system design D, graphically represented by the dashed arrows in Figure 2.4. Depending on the considered system hierarchy level, the integrated system realization can be, for example, an integrated set of software components, a test bench with some but not all hardware components, or a complete (prototype of a) system. Testing involves defining tests for the considered aspects of the system, executing these tests on the system realization, and determining the test outcome (pass or fail) by comparing the test results to the requirements R and the design D. During the execution of a test, the test conditions of the system are influenced by providing test inputs or stimuli, and the resulting system behavior is observed via test outputs, which should conform to the expected test outputs. Different types of tests exist, ranging from functional tests to performance tests, as well as regression and acceptance tests, which are described in more detail in Chapter 6. A common property of these tests is that they require realized and integrated components. This implies that these tests can only be performed in the I&T phases and, when problems are detected and need to be fixed, the effort invested in these phases increases and timely shipment of the system is directly threatened.

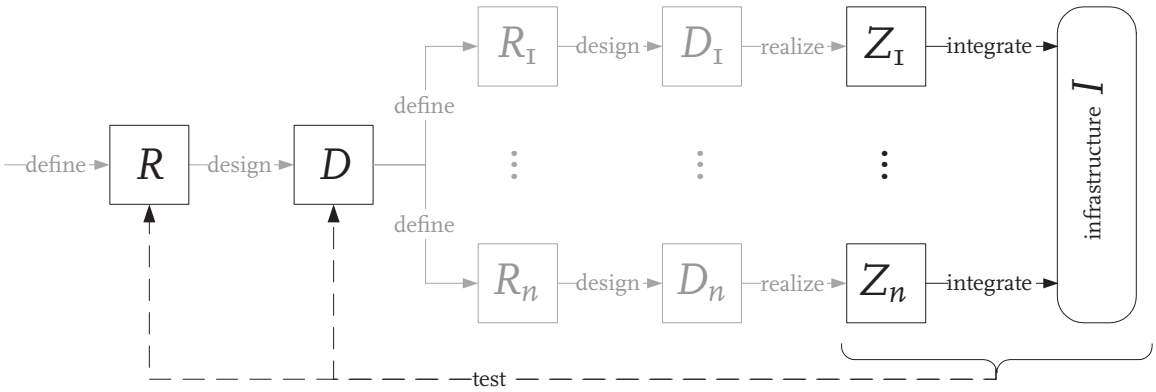


Figure 2.4: Integration and system testing

2.2 Model-based engineering

As previously mentioned, the current system development process is mostly based on documents, which has several disadvantages. First, practical experience shows that documents may be scattered, incomplete, or outdated, and that they may contain information that is ambiguous or inconsistent, usually with more focus on the nominal behavior than on the exceptional behavior. As a result, it is difficult to obtain a good system overview and to detect inconsistencies and potential problems based on documents only. Second, the informal structure of documentation complicates automated analysis techniques such as inconsistency detection, which currently leaves manual document reviewing as the main technique used for document analysis. Third, documentation is a static piece of information which makes it difficult to express and analyze dynamic system behavior. As opposed to that, *executable* specifications [Fuchs 1992] enable a more thorough and systematic analysis of dynamic system behavior. Fourth, determining the integrated system behavior based on component documentation only is a difficult task that requires a considerable amount of component design knowledge as well as system overview. In other words, using documentation to represent a component is not suitable for early integration and testing, which leaves the (later available) realization as the only representation of a component that is suitable for I&T activities.

An emerging alternative to document-based system development is to use models to represent the components of a system, and to use a range of model-based techniques and tools to support the system development process. In general, we consider a model to be an abstract representation of a real component or system, used in experiments to gain knowledge about the real component or system. Although different types of models can be used, e.g., scale models of cars to analyze aesthetics as well as aerodynamics, we particularly focus on computer models describing the behavior of components and systems. Such models have several advantages over documentation. First, they provide a structured and systematic approach to specify component and system behavior with more consistency and less ambiguity than documents, because the model semantics precisely defines what a certain modeling construct means. Second, well-defined model semantics also allows automatic reasoning by

tools, enabling model execution and the use of various sophisticated and automated analysis techniques. Third, executable models make it easier to analyze dynamic behavior as well as the performance of a component or a system. Fourth, when a modeling language supports model composition, integrating component models and observing the emergent system behavior becomes as easy as modeling and analyzing a component model only. These advantages show that, as an alternative to the component realization, also a model is a representation of a component that is suitable for I&T activities. In the case that models of the components are available before the realizations, the models can effectively be used for early integration and testing.

Many fields of academic research investigate and develop model-based techniques and tools that support a reduction of effort invested in particular phases of the system development process. Although the general approach of using models to support the system development process is not an established research field as such, we refer to it as the *model-based engineering* approach.

Examples of model-based engineering techniques and tools are model-based requirements engineering [Von der Beeck et al. 2002], model-based design [Gomaa 2000; The Mathworks – SIMULINK 2007], model checking [Katoen 1999], model-based code generation [De Wulf et al. 2005; Huang et al. 2006], hardware/software co-simulation [Rowson 1994; PTOLEMY project 2007], hardware-in-the-loop testing and rapid prototyping [Hanselmann 1996; Deppe et al. 2004], and model-based testing [Brinksma and Tretmans 2001; Hartman 2002]. In most cases, however, these model-based engineering techniques are investigated in isolation and only cover specific phases of the system development process, and little work is reported on combining such techniques into an overall model-based engineering method. Although model-based systems engineering [Ogren 2000] and the Model-Driven Architecture [Kleppe et al. 2003], which uses the Unified Modeling Language (UML) [UML 2007] for software development, are such overall model-based engineering methods, they mainly focus on the requirements, design, and implementation phases, rather than on the I&T phases. Furthermore, literature barely mentions realistic industrial applications of such overall methods, at least not for high-tech multi-disciplinary systems.

In the remainder of this chapter, the model-based engineering approach is used to answer the research questions related to early integration and testing. Since a wide range of methods, techniques, and tools related to model-based engineering are already available and still being developed in other research projects, we have the intention to reuse these results as much as possible, instead of developing new methods, techniques, and tools. Where necessary, the available methods, techniques and tools will be adapted, extended, and combined to accommodate the requirements for early integration and testing.

2.3 Model-based integration and testing method

This section proposes several model-based I&T techniques as answers to QUESTIONS 1.1 and 1.2. The model-based I&T techniques are inspired by the model-based engineering approach as introduced in the previous section.

2.3.1 System analysis

This subsection provides an answer to QUESTION 1.1, which is repeated below.

> **QUESTION 1.1** How can system risk be revealed and subsequently reduced when no component realizations are available?

Many model-based techniques that are used during the system development process only consider models of the *intended* system behavior, i.e., how the system should work. The goal of these techniques is to guide the activities in the system development process such that the correctness of the system model, i.e., the intended system behavior, is preserved throughout the process. In this top-down approach, the abstract system model is continuously refined using correctness preserving model transformations, ultimately resulting in realizations that are synthesized from the models. However, applying this in the current document-based way of working would imply many changes to the system development process, which is not considered in this Ph.D. project.

Nevertheless, model-based techniques can still support the current I&T process by using a bottom-up approach instead of a top-down approach. This can be achieved by using models as representations of the *actual* behavior of the components, i.e., as they are designed and realized in the current, unchanged way of working. The integration of these component models yields a system model that is an early representation of the actual system behavior. This implies that also the corresponding system risk is revealed before the components themselves are realized and integrated, which answers the first part of QUESTION 1.1 on revealing risk. In contrast to the above mentioned top-down model-based approaches that aim at preserving *correct* system behavior, the integration of models that contain the actual component behavior may often result in *incorrect* system model behavior, e.g., due to an unforeseen conflict in the component designs. This bottom-up approach perfectly matches the main goal of integration and testing, namely detecting system level problems that emerge from the integrated components as early as possible.

The following requirements need to be satisfied when models of components are used to reveal system risk early, such that it can be reduced before the components are realized.

Information sources for modeling The modeling activities require sufficient sources of information on what should be modeled, e.g., the model boundary with inputs and outputs, the assumptions of the model, and the internal model behavior. In this case, the models should represent the components as they are designed, so the component designs themselves are chosen as the main information sources for modeling. The available design documents are taken as a starting point for modeling, and, if they do not provide sufficient information, discussions with the involved engineers are used to clarify the designs. In this way, the discussions during the modeling activities also help to detect problems, inconsistencies, and potential issues in the design documentation and to increase the system overview for the engineers.

Model expressivity To be able to represent components by models, the modeling language that is used should offer sufficient expressivity to describe the considered aspects of the

considered components. Modeling different aspects of different components may require different modeling language paradigms and corresponding techniques and tools, which is described in more detail in Section 2.4.

Model compositionality Integrating component models such that a system model is obtained, requires a modeling language that supports compositionality. This means that modeling constructs should be provided that can be used to couple different component models, similar to the infrastructure *I* in Figure 2.2 that is used to couple different component realizations.

Model correctness When models are replacing the realizations of components to detect problems by early I&T activities, it is essential that the models are correct representations of the components as they are designed (and subsequently realized). This means that the correctness of a model with respect to the component as it is designed should be analyzed. Since the component designs are captured in (informal) documents, it is difficult to define a strict correctness relation between the model and the design documentation. In practice, manually comparing the model against the design documentation and discussing the model with the involved engineers (possibly supported by model execution) can be used to improve the confidence in the model. This model validation process continues until there is sufficient confidence that the model is a correct representation of the component and therefore a suitable replacement of its realization to be used for system testing.

The second part of QUESTION 1.1 concerns reducing the system risk that is revealed early by using models as described above. To achieve this, the behavior of the integrated system model should be analyzed to check whether it satisfies the system requirements *R* and whether it corresponds to the intended system design *D*. When the system model behavior is found to be correct with respect to *R* and *D*, the probability of faults that cause incorrectness decreases, i.e., the corresponding system risk is directly reduced. When the system model behavior is found to be incorrect with respect to *R* and *D*, a problem is detected which can be fixed at an early stage, i.e., the corresponding system risk is indirectly reduced. This means that model-based techniques for system analysis enable a reduction of system risk before the component realizations are available and integrated, which completes the answer to QUESTION 1.1.

In order to analyze and test the integrated system model such that system risk can be reduced at an early stage, the following requirements need to be satisfied.

Determine model behavior Techniques are required to interpret the modeling constructs used to express the component designs, from which the resulting behavior of the system can be determined.

Analyze model behavior Besides techniques to determine the system model behavior, other techniques and tools are required to analyze this system model behavior. For example, the determined system model behavior should be executed and the executed behavior should be presented in a form that can easily be compared to the intended system behavior, or the determined system model behavior should be checked against certain required properties.

Figure 2.5 shows the proposed model-based I&T technique of creating and integrating component models such that the resulting system model can be analyzed against the system requirements R and system design D. Each component $C_i \in C_{1..n}$ is represented by a model M_i that is based on the design D_i. The component models $M_{1..n}$ are integrated using a model of the infrastructure I_M, yielding the integrated system model $\{M_{1..n}\}_{I_M}$. The integrated system model $\{M_{1..n}\}_{I_M}$ is analyzed against the system requirements R and the system design D (graphically represented by the dashed arrows) by deriving properties and scenarios from R and D, for which the correctness is determined using various model-based analysis techniques, which is explained in more detail in Chapter 3. Note that the infrastructure can be modeled as I_M on different levels of abstraction, ranging from completely ignored infrastructure, e.g., when infrastructure details are unknown or irrelevant for system analysis, up to detailed infrastructure models, e.g., when details of the component interaction are important for system analysis. This is explained in more detail in Chapter 5.

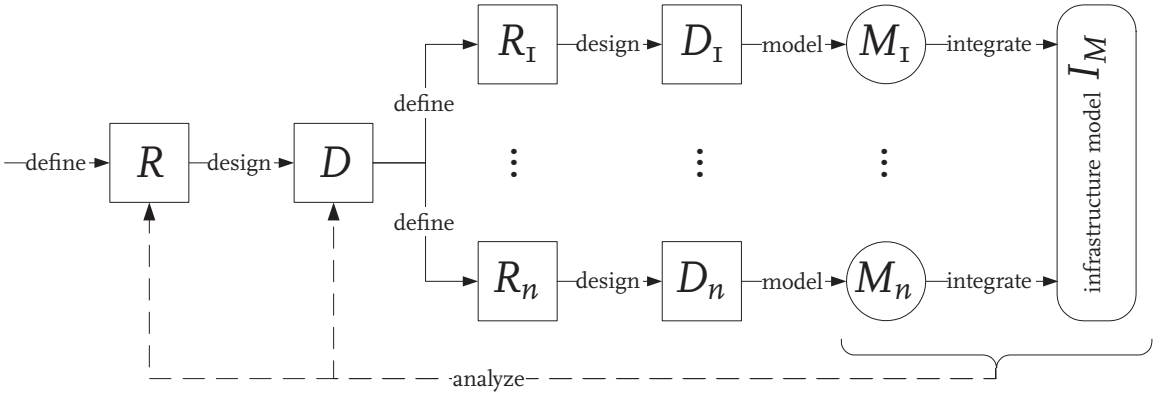


Figure 2.5: Model-based system analysis

Besides model-based system analysis, in which only models are considered for the analysis, it would also be interesting to analyze and test the system when only some component realizations are available, as expressed in QUESTION 1.2. Before answering QUESTION 1.2, however, another model-based technique is introduced that focuses on testing component realizations on the component level rather than on the system level.

2.3.2 Component testing

In current industrial practice, when a component realization Z_i is available, it is tested against its requirements R_i and its design D_i. As previously mentioned, testing involves defining tests for the considered aspects of the component, executing these tests on the component realization, and determining the test outcome (pass or fail) by comparing the test results to the requirements R_i and the design D_i. Currently, most of these test activities are performed manually, e.g., defining tests based on documents that specify the requirements R_i and the design D_i, which makes current component testing a tedious and time consuming task. When a component is represented by a model M_i for which the correctness has been shown, e.g., using the model-based analysis techniques presented in the previous subsection, the

model can also be used as a reference for component testing.

In the model-based testing research field [Brinksma and Tretmans 2001], which is the topic of another part of the TANGRAM project, theories, techniques and tools are developed that use models for automatic testing of component realizations. Using a test generation algorithm, tests are automatically generated from the model and subsequently executed on the component realization, which means that more tests can be executed per time unit. Although this model-based technique does not directly answer one of the QUESTIONS in this thesis because it does not focus on system risk, more tests per time unit is a test phase improvement as proposed by the second solution in Section 1.4, which does contribute to the main objective of reducing the disadvantageous influence of the I&T phases on the T-Q-C business drivers. Therefore, model-based component testing is included in the proposed method of using model-based techniques for early integration and testing.

The following requirements need to be satisfied for model-based component testing.

Suitable test generation algorithm A test generation algorithm should be available that is suitable for automatic model-based testing of the considered aspects of the considered components. If the test generation algorithm requires a certain modeling language as input, this language should offer sufficient expressivity to describe these aspects.

Test tool with connection to realization A test tool is required that implements the test generation algorithm and contains functionality for automatic test execution. Furthermore, the test tool should provide easy access to the considered interfaces of the realization under test, i.e., the interfaces to which test inputs are provided and from which test outputs are observed.

Figure 2.6 contains a graphical representation of model-based component testing in the context of the development process of a component C_i (with R_i, D_i, and Z_i, corresponding to Figure 2.2). The figure shows that, based on the design D_i, a model M_i is created (corresponding to Figure 2.5). Using a model-based test tool, tests are automatically generated from model M_i and executed on the component realization Z_i to test whether Z_i conforms to M_i, which is graphically represented by the dashed arrows.

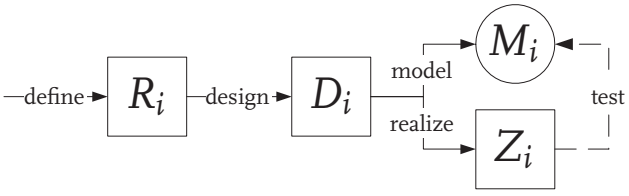


Figure 2.6: Model-based component testing

2.3.3 Integration and system testing

This subsection provides answers to QUESTION 1.2, which is repeated below.

> **QUESTION 1.2** How can system risk be revealed and subsequently reduced when only some component realizations are available?

In the case that only some component realizations are available, i.e., without a complete system, it is impossible to perform system tests in the current way of working. However, in the case that components are represented by models, these models can replace component realizations that are not yet available. By integrating models with available component realizations, a *model-based integrated system* is obtained, i.e., an early representation of the system realization in which the corresponding system risk is revealed before all components realizations are available. When the integrated models and realizations can be executed together, this model-based integrated system can be used for early system testing and corresponding risk reduction, thus answering QUESTION 1.2.

Since model-based integration and system testing does not require that all component realizations are available, and since models can usually be available earlier than realizations, this model-based I&T technique enables earlier detection and prevention of problems when compared to real system testing. Furthermore, models usually allow easier adaptation and configuration than realizations, which means that they are well suited for system testing under different conditions. Especially for exceptional behavior testing, creating the non-nominal test conditions, e.g., a broken component, is usually easier and less expensive when models are used instead of realizations. Besides that this improves the coverage and thus the quality of tests, the ability to rapidly change test conditions using models also improves the efficiency of test execution. As such, model-based system testing not only allows earlier testing, but it also increases the risk reduction rate for the test phase, as proposed by the second solution in Section 1.4.

In order to integrate models and realizations and to perform system tests on the resulting model-based integrated system, the following requirements need to be satisfied.

Connect models and realizations When models and realizations are integrated, they need to interact and communicate with each other. Communication actions specified in a model are normally interpreted and executed in a model environment only. However, to establish a connection between the model of a component and realizations of other components, these communication actions in the model have to be transformed into real communication actions. For example, a send action in the model needs to be transformed into a real message that is sent to other components, while real messages sent to the model need to be transformed into receive actions in the model itself.

Execute models in realization environment Since a model-based integrated system includes at least some realizations, tests have to be executed in the realization environment, which means that also the model behavior should be executed in the realization environment, i.e., in *real-time*. This means that all actions, e.g., the physical actions that are modeled, but also calculations and communication actions, take a certain amount of time to be executed, i.e., the real-time behavior corresponds to the model time behavior. For example, when a model specifies a movement from position A to position B which takes ten seconds, then the real-time execution of this behavior consists of updating the position variable from A to B and performing a real-time delay of ten seconds. When the same model would be simulated in a model environment using model time, the execution of this behavior would consist of updating the position variable from A to B and adding ten seconds to the variable that represents the time in

the simulation model. In real-time, it takes only a small amount of time, e.g., a few milliseconds, to perform these variable updates, i.e., the real-time behavior does not correspond to the time behavior in the simulation model.

Determine and analyze model-based integrated system behavior Similar to system analysis, techniques are required to determine and analyze the behavior of a model-based integrated system, e.g., by using the same tests as for the system realization. As previously mentioned, different configurations of the models can be used to test the system behavior under different (non-nominal) conditions.

As shown in Figure 2.2, the infrastructure I establishes the interaction between the realizations of the components. For successful interaction between two component realizations, they must use the same interaction type, e.g., function calls or message passing. However, models of components may use interaction types that differ from those used by realizations, e.g., more abstract interaction types to reduce the complexity of the model. For the integration of models and realizations, this means that two interacting components may use different interaction types, which requires an infrastructure I that enables a connection between the involved interaction types. This infrastructure is referred to as the model-based integration infrastructure, denoted by I_{MZ}, and is discussed in more detail in Chapter 5.

Figure 2.7 contains a graphical representation of model-based integration and system testing. When only the depicted components C_1 and C_n are considered, the figure shows that component C_1 is represented by model M_1 (corresponding to Figure 2.5), while component C_n is represented by realization Z_n (corresponding to Figure 2.2). Using the model-based integration infrastructure I_{MZ}, model M_1 and realization Z_n are integrated, yielding the model-based integrated system $\{M_1, Z_n\}_{I_{MZ}}$. This early representation of the system is then tested on the system level using tests derived from the system requirements R and the system design D, which is graphically represented by the dashed arrow.

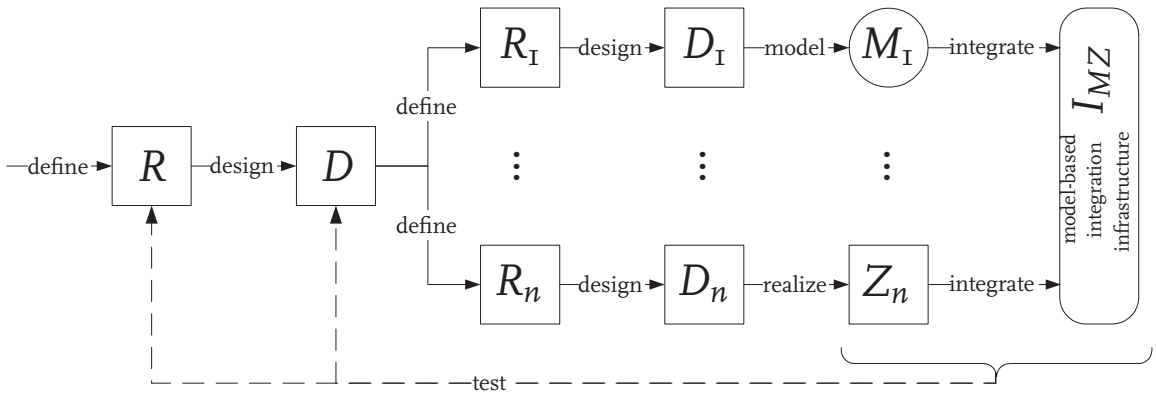


Figure 2.7: Model-based integration and system testing

2.3.4 MBI&T method

The previous subsections proposed several model-based I&T techniques as answers to QUESTIONS 1.1 and 1.2. These techniques were presented individually without explicit relations to each other or to the current system development process from Section 2.1. These relations can easily be identified by combining the previous figures of this chapter, i.e., Figure 2.2 showing the current system development process, Figure 2.5 showing model-based system analysis, Figure 2.6 showing model-based component testing, and Figure 2.7 showing model-based integration and system testing. The result, which is referred to as the *model-based integration and testing method* or MBI&T method is shown in Figure 2.8, leaving out the dashed arrows of the previous figures that depicted the different analysis and test activities. Similar to the current system development process of Figure 2.2, this figure shows a 'sequential' development process for simplicity, abstracting from the parallelism, increments and iterations that exist in practice. The development process of each component C_i (with requirements R_i and design D_i) results in a realization Z_i as well as in a model M_i. When the system is integrated, each component C_i is represented by either model M_i or realization Z_i, which is graphically represented by the integration 'switches'. For example, the positions of the integration switches in Figure 2.8 imply that model M_1 and realization Z_n are integrated, i.e., $\{M_1, Z_n\}_I$. Note that with code generation [De Wulf et al. 2005; Huang et al. 2006], the realization of a software component Z_i could also be synthesized from its model M_i, however this is not further considered in this thesis.

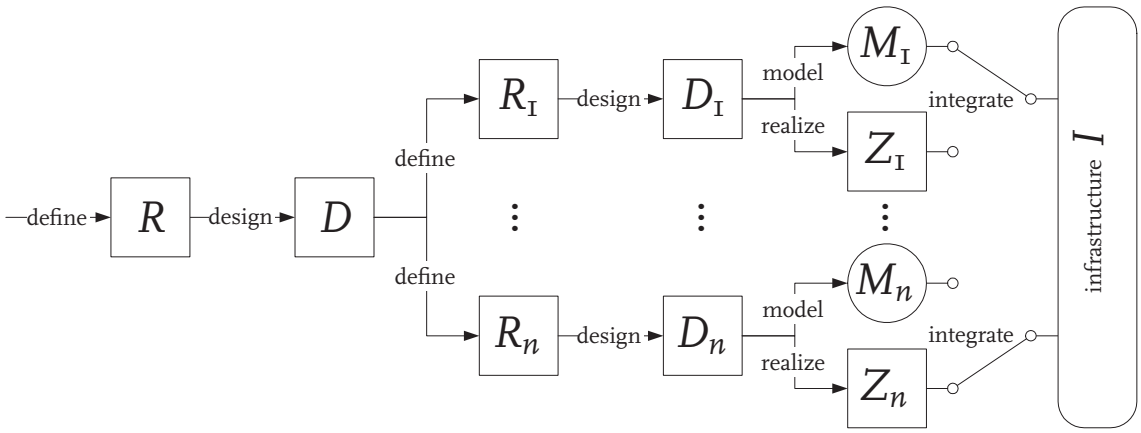


Figure 2.8: System development process in the MBI&T method

Depending on the component representations that are integrated, i.e., only models, combined models and realizations, or only realizations, a different type of infrastructure I may be required. When only models are integrated, the infrastructure model I_M as shown in Figure 2.5 is used. When combined models and realizations are integrated, a model-based integration infrastructure I_{MZ} as shown in Figure 2.7 is used. Finally, when only realizations are integrated, which is also the case for the current system development process as shown in Figure 2.2, the infrastructure realization, denoted with I_Z, is used. Chapter 5 gives a more detailed description of these different types of infrastructure and their usage in the

MBI&T method. Figure 2.8 abstracts from these different types of infrastructure, and uses the 'generalized' infrastructure I. Note that the infrastructure I could also be considered as a component C_i in the MBI&T method. This would mean that it can only exist as infrastructure model I_M or as infrastructure realization I_Z, similar to M_i and Z_i for a component C_i. However, when the infrastructure is used to integrate models and realizations, it has to be able to connect these different component representations, i.e., it should include some parts of I_M to connect models and some parts of I_Z to connect realizations. Because of this 'intermediate' type of infrastructure, i.e., the model-based integration infrastructure I_{MZ}, infrastructure I is distinguished from the components $C_{1..n}$ in the MBI&T method. Nevertheless, similar to the components $C_{1..n}$, the infrastructure I also has requirements and a design, which are usually part of the system requirements R and system design D.

The following procedure summarizes all activities of the MBI&T method, which are referred to as MBI&T activities. The procedure considers a system with n components, and takes the component designs D_i and the infrastructure design, which is part of system design D, as a starting point.

1. **Modeling**
   (a) Modeling of the components M_i based on D_i.
   (b) Modeling of the infrastructure I_M based on D, possibly on different abstraction levels, depending on the availability of the infrastructure design and on the goal of the analysis in activity 2b.
2. **System analysis**
   (a) Integration of component models $M_{1..n}$ using the infrastructure model I_M, resulting in the system model $\{M_{1..n}\}_{I_M}$.
   (b) Analysis of system model $\{M_{1..n}\}_{I_M}$ using various model-based techniques.
3. **Integration and system testing**
   In the system model $\{M_{1..n}\}_{I_M}$, the infrastructure model I_M is replaced by the model-based integration infrastructure I_{MZ}, yielding $\{M_{1..n}\}_{I_{MZ}}$. Then, for each realized component $Z_i \in Z_{1..n}$ that becomes available:
   (a) Testing of component realization Z_i against M_i using automatic model-based testing.
   (b) Removal of model M_i from the system $\{\ldots, M_i, \ldots\}_{I_{MZ}}$ and integration of realization Z_i as its replacement, yielding $\{\ldots, Z_i, \ldots\}_{I_{MZ}}$. This activity corresponds to 'flipping' the integration switch of component C_i from M_i to Z_i in Figure 2.8.
   (c) Testing of the integrated system obtained in activity 3b, using tests derived from system requirements R and system design D.

Note that these MBI&T activities are complementary to the current way of working and do not require changes in the current system development process. However, the activities do require a certain amount of time to be invested by the involved engineers for answering

questions about unclarities in their designs and for discussing the models and the analysis results. In return, the MBI&T activities provide them with valuable feedback on the system behavior and increase their system overview. In addition to these time investments, a MBI&T activity that involves component realizations requires that test resources, e.g., a component realization or a prototype system, are allocated to perform the MBI&T activity. This may influence the current I&T process in a sense that there are more possible I&T activities to which the (limited) test resources must be allocated. This trade-off between the costs and the benefits of the MBI&T activities is discussed in Chapter 6, which also proposes a quantitative method to decide where (QUESTION 2.1) and when (QUESTION 2.2) the I&T process can profit from using models.

2.4 Method instantiation

The previous section presented the MBI&T method in a generic way, without explicit notion of which components and aspects are considered, which types of models and infrastructure are used, and which analysis and test techniques are applied. Although this generic MBI&T method was sufficient to explain how QUESTIONS 1.1 and 1.2 are answered, it is not sufficient to answer QUESTION 1.3, repeated below, on the applicability and profitability of the method in current industrial practice.

> **QUESTION 1.3** Is it feasible and profitable to apply the proposed early I&T method in current industrial practice?

This research question is answered by applying the proposed MBI&T method to the EUV case study introduced in Chapter 1. However, before the MBI&T method can be applied to any system, it needs to be *instantiated*. This means that the generic MBI&T method of the previous section is 'filled in' with concrete components and aspects considered, with types of models and infrastructure to be used, and with suitable analysis and test techniques to achieve the objectives of the MBI&T activities, i.e., revealing and reducing system risk at an early stage.

2.4.1 Paradigm, mathematics, tools

An instantiation of the MBI&T method consists of a *paradigm* together with related *mathematical techniques* and *tools*. A paradigm is "a set of assumptions, concepts, values, and practices that constitutes a way of viewing reality for the community that shares them, especially in an intellectual discipline" [American Heritage Dictionary of the English Language 2007]. In the context of the MBI&T method, a paradigm provides a way of thinking and reasoning about a system, as well as a way of modeling and analyzing the system. The suitability of a particular paradigm for an instantiation of the MBI&T method depends on the considered aspects of the system (often related to particular engineering disciplines) and on the objectives of the MBI&T activities.

In order to actually perform MBI&T activities in a particular paradigm, suitable and adequate mathematical techniques for modeling and analysis are required, which should be implemented in tools to be useful in practice. These techniques and tools should satisfy the requirements of the MBI&T activities described in Section 2.3. Research related to different paradigms focus on different modeling and analysis techniques, which means that the maturity and applicability of particular techniques and the availability of practical tools varies per paradigm. An example of using different paradigms, mathematics, and tools for model-based analysis of an industrial system is given in [Braspenning et al. 2006a].

A main characteristic to differentiate between paradigms and corresponding mathematics is the way they consider time behavior [Van Beek and Rooda 2000]. Many systems have properties that vary continuously in space and/or time, e.g., mechanical, electrical, and physical systems. Such systems are usually modeled by describing their *continuous-time* behavior in the form of differential equations. Other systems have properties that change only at discrete points in time (events), e.g., operation research, supervisory machine control, and real-time systems. Such systems are usually modeled by describing their *discrete-event* behavior in the form of states and events, e.g., in state machines or petri nets. To bridge the gap between these different concepts of describing time behavior, *discrete-time* models or *hybrid* models can be used. Discrete-time models are often used in digital applications, e.g., digital control of continuous-time systems, in which the continuous-time behavior is sampled and the state changes are described at equidistant points in time using difference equations. As such, a discrete-time model can also be considered as a special case of a discrete-event model in which the events are equidistantly spaced in time. Finally, several modeling languages combine continuous-time and discrete-event concepts to model hybrid systems such as chemical process plants, see [Van Beek and Rooda 2000] for an overview.

The next paragraphs describe different model-based analysis techniques and give a flavor of different paradigms in which these techniques are used. Some paradigms use models that describe only one possible behavior, which can be calculated based on a set of equations. For example, models in the dynamics and control paradigm are often based on differential equations, which can be used to exactly calculate, amongst others, trajectories, settling times, and eigenfrequencies [Franklin et al. 2005]. Models in the structural mechanics paradigm and in the fluid dynamics paradigm are usually based on partial differential equations, for which an approximate solution, e.g., deformation and stress in structures or velocity and pressure in fluids, can be determined using numerical techniques such as the finite element method (FEM) [Chandrupatla and Belegundu 2002].

Another common and widely available technique to analyze the behavior of a system before it is realized is *simulation*, in which one possible model behavior is determined. For example, in the electronics paradigm, circuit simulators [SPICE 2007] are able to translate the elements and connections of a circuit into differential algebraic equations, which can be used, for example, to determine and optimize the power consumption of the circuit. Models may describe many possible behaviors, especially when they include stochastic elements, e.g., noise in a dynamics and control model or different economical scenarios in a business process model to estimate economical risks and expected profits, or when they include parallelism and non-determinism, e.g., models of software components and their interaction in the concurrent process paradigm. In cases where analytical techniques, e.g., using Markov

chains [Tijms 2003], cannot be applied, multiple simulation runs, e.g., using Monte Carlo techniques [Liu 2001], may be used to deal with the uncertainty in the analysis results.

In most cases, simulation is not exhaustive, i.e., not all possible behaviors are analyzed, which means that simulation can only show that a model *might* have correct behavior. Especially in the concurrent processes paradigm, proving the correctness of a model in general is important, e.g., to detect and avoid deadlocks in software. Therefore, a lot of research in this paradigm focuses on formal verification techniques such as *model checking* [Katoen 1999], an analysis technique to determine and check all possible model behaviors against a specified property. Also other paradigms are investigating model checking as a potential analysis technique, e.g., by extracting formal models from informal models in the dynamics and control paradigm [The Mathworks – SIMULINK design verifier 2007].

In contrast to the widely applied analysis techniques mentioned above, which focus on models only, model-based integration techniques that combine models and realizations are less common for many paradigms. In the dynamics and control paradigm, e.g., in automotive and mechatronics applications, hardware-in-the-loop testing (controller realization with plant model) and rapid prototyping (controller model with plant realization) are frequently used [Hanselmann 1996; Deppe et al. 2004; Verhoef et al. 2007]. Also applications in the real-time embedded software paradigm often use hardware/software co-simulation techniques, in which the software realization is executed on a model of the hardware platform [Rowson 1994; PTOLEMY project 2007]. For other paradigms such as the concurrent processes paradigm, however, less attention is paid to the analysis and testing of combined models and realizations.

In general, most of the example instantiations mentioned above are useful for modeling and analyzing mono-disciplinary components or systems for which all components can be expressed in the same paradigm. Since the MBI&T method focuses on the analysis and testing of multi-disciplinary systems, an instantiation should be capable of dealing with multiple paradigms. Several approaches can be used to combine multiple paradigms or disciplines in one analysis method, e.g., the *glued* approach or the *integrated* approach as described in [Saleh and Newton 1990; Fleurkens 1996]. In the glued approach, the components are modeled using the most appropriate and therefore often different paradigms, mathematics, and tools. Analysis of the system behavior is possible by coupling (parts of) different analysis tools, either on the execution engine level, i.e., the coupled execution engines negotiate a priori which actions will be executed, or on the executed behavior level, i.e., the execution engines react a posteriori on the model behavior as executed by other execution engines. These couplings involve a mapping and conversion of communication signals and a synchronized time advancement mechanism, usually implemented in so-called 'glue' software. Examples of this glued approach are couplings between discrete-event models of real-time software controllers and continuous-time models of the controlled machine dynamics [Hooman et al. 2004; Verhoef et al. 2007], and couplings between electronic circuit simulators and FEM tools for electro-thermal analysis [Wünsche 1996]. In many cases, this approach can only be applied to a subset of the involved paradigms, which means a reduction of the modeling expressivity.

In the integrated approach, the behavior of the components from different disciplines is expressed in one set of mathematics and tools, i.e., coupled on the model level, which can be

done in two ways. The first type of the integrated approach requires one set of mathematics and tools that supports and combines multiple paradigms. A major problem in this approach is to obtain this one set of mathematics such that the different semantics of all supported paradigms and mathematics are combined and integrated in a consistent way, e.g., by using formal model transformations or a protocol that synchronizes certain actions. Examples based on this type of the integrated approach are STATEFLOW [The Mathworks – STATEFLOW 2007], which extends the dataflow block diagrams (for modeling continuous-time behavior) from SIMULINK [The Mathworks – SIMULINK 2007] with statecharts (for modeling discrete-event behavior), and PTOLEMY [PTOLEMY project 2007], a software platform that focuses on heterogeneous mixtures of different models of computation to design hybrid embedded systems.

The second type of the integrated approach uses one paradigm in which equivalents of other paradigms can be expressed. This approach requires that the chosen paradigm is supported by mathematics and tools that have sufficient expressivity to model the equivalents of the other paradigms, at least for those parts that are significant for the system behavior to be analyzed. Examples of the second type of the integrated approach are budget-based modeling [Freriks et al. 2006], in which design choices for individual components (from different disciplines) are related to the system performance parameters using simple equations; SYSML [SYSML 2007], an adaptation of the software-focussed UML [UML 2007] to model systems engineering applications including non-software components such as hardware, processes, and personnel; MODELICA [MODELICA project 2007], which contains libraries to express components from many different engineering disciplines as differential and algebraic equations; and process algebraic languages such as χ (Chi) [Van Beek et al. 2006a] whose expressivity allows for (abstract) modeling of concurrent processes as well as various physical phenomena.

The examples of the glued and integrated approach mentioned above show that there are many developments in the area of multi-disciplinary system modeling and analysis. However, in most cases, not all semantics of the supported paradigms have been formalized. Furthermore, most of the mentioned examples do not support all proposed activities of the MBI&T method. As described in the following subsection, we investigate whether the process algebraic language χ, which does have a complete formal semantics, can be used to perform all MBI&T activities in current industrial practice.

2.4.2 MBI&T instantiation in this thesis

In this thesis, the MBI&T method is instantiated with the paradigm of *concurrent processes*, together with mathematical techniques and tools based on *process algebra* [Baeten and Weijland 1990; Fokkink 2000]. Our hypothesis is that this instantiation fulfills all requirements of the proposed MBI&T activities and that it is suitable to perform these activities in industrial practice, focusing on component interaction and time behavior. In the remainder of this thesis, this hypothesis is tested by applying the instantiated MBI&T method to the EUV case study introduced in Chapter 1, which answers QUESTION 1.3.

In the concurrent processes paradigm, system behavior is considered to be composed of several dynamic and active processes that are executed concurrently, where these processes

exchange static and passive data in order to influence each other's behavior [Fokkink 2000]. Concurrent processes are well suited to describe and reason about component interaction and time behavior, which are important aspects of the emergent system behavior when integrating and testing high-tech multi-disciplinary systems. Practical experience shows that designing and analyzing component interaction and time behavior is a difficult and error-prone task, resulting in a significant system risk that, in the current way of working, is difficult to reduce without having an integrated system realization available. This means that the potential benefits of a MBI&T method that focuses on interaction behavior and time behavior are significant. Furthermore, there is not yet an established industrial way of working for modeling and analysis in the concurrent processes paradigm, at least not in a sense as SIMULINK for the dynamics and control paradigm. This absence of an established industrial way of working makes it an interesting and promising area to investigate.

The mathematics and tools used in the remainder of this thesis are based on χ (Chi), a process algebraic language with a complete formal semantics, developed at the Eindhoven University of Technology [Van Beek et al. 2006a; Man and Schiffelers 2006; Baeten et al. 2007]. The χ language and the corresponding toolset [Systems Engineering Group 2007; Hofkamp and Rooda 2007] are used as a carrier to show the practical applicability and the profitability of the MBI&T method. The χ language is intended for modeling, simulation, verification, and real-time control of discrete-event, continuous-time, or combined, i.e., hybrid, systems. Examples of industrial systems that have successfully been modeled in χ include integrated circuit manufacturing plants [Haagh et al. 1998], process industry plants [Van Beek et al. 2002], and machine control systems [Van de Mortel-Fronczak et al. 2001]. The χ toolset allows modeling and simulation of χ models, as well as translation of χ models to other formalisms that enable the use of model checking tools such as UPPAAL [UPPAAL 2007] and SPIN [SPIN 2007]. Real-time and distributed execution of χ models, optionally in combination with other non-χ components, is possible by using a real-time simulator and a suitable middleware solution. This thesis also investigates the possibility of using χ models for automatic model-based component testing with the test tool TORX [TORX 2007], which was also used in another part of the TANGRAM project. The following chapters of this thesis describe in more detail how these χ model-based techniques were used to perform the MBI&T activities.

The χ modeling language has a consistent equations semantics that supports differential algebraic equations, functions, and calculations on variables of different data types. This enables the modeling of various physical phenomena such as motion, flow, and friction, as well as electronic circuits, supervisory machine control, and process plants, as shown in [Man and Schiffelers 2006]. This corresponds to the second type of the integrated approach as described in Subsection 2.4.1: although the χ language focuses on the concurrent processes paradigm, it is sufficiently expressive to model equivalents of other paradigms as well. In the instantiated MBI&T method, however, the focus is on early analysis and testing of component interaction and time behavior, which is not directly influenced by details of the internal component behavior such as the physical phenomena mentioned above. To analyze or test the component interaction and the emerging system behavior, these details can safely be abstracted, e.g., by expressing only relevant action durations and parameter changes. Taking motion as an example, parameters such as position, speed and acceleration could in prin-

ciple be calculated in detail by using the supported features of χ. In many cases, however, other components are not influenced by the motion itself, but only by relevant positions and durations of movements between these relevant positions, e.g., a sensor that detects a certain position of the component. When these relevant positions and durations are known, e.g., calculated using domain-specific paradigms, mathematics, and tools, they can be used to model the motion behavior in an abstract but equivalent way.

2.5 Conclusions

This chapter started with a description of the current system development process, which was improved by applying a range of model-based engineering techniques particularly focusing on early integration and testing. Several MBI&T activities were proposed as answers to QUESTIONS 1.1 and 1.2. Model-based system analysis involves the modeling of each component, the integration of the component models, and the analysis of the integrated system model. Model-based component testing involves automatic generation of tests from a component model, as well as automatic execution of the generated tests on the component realization. Model-based integration and system testing involves the integration of realized components with models of not yet realized components, and subsequent testing of this model-based integrated system. Each of these MBI&T activities has several requirements for the models and for the analysis techniques to be used.

Combining these MBI&T activities with the current system development process yielded the MBI&T method as shown in Figure 2.8. Before this method can be applied in practice, it needs to be instantiated with a paradigm, mathematics, and tools. In this thesis, the MBI&T method is instantiated with the concurrent processes paradigm, together with mathematics and tools based on process algebra. In particular, the process algebraic language χ and its toolset are used as carrier to show the applicability and the profitability of the MBI&T method in current industrial practice, as expressed in QUESTION 1.3. Our hypothesis is that this instantiation fulfills all requirements of the proposed MBI&T activities and that it is suitable to perform these activities in industrial practice, focusing on component interaction and time behavior. In the next three chapters, the acceptability of this hypothesis is tested by applying the instantiated MBI&T method to the EUV case study introduced in Chapter 1, answering QUESTION 1.3.

CHAPTER 3

System analysis

How to analyze a system without a system?

It is in the nature of all people, especially those who invent and engineer new things, to see whether their ideas really work in practice. Engineers involved in the development of a high-tech multi-disciplinary system want to know whether the system, with all their ideas integrated in it, will work, since this provides them confidence that they are on the right track. They also want to know as soon as possible whether the system will *not* work, since this gives them early feedback on the problems and gives them more time to solve these problems. In both cases, some form of system analysis is required to determine whether the system will work as expected or which problems need to be solved in the system.

In the current system development process, engineers usually know reasonably well what the system should do (based on requirements R) and what they expect that the system will do (based on system design D and component designs $D_{1..n}$). In contrast to this *intended* system behavior, it is often more difficult to show the *actual* system behavior before the system realization is available, i.e., before all components of the system are realized and integrated. Analyzing the actual system behavior without a realized system requires significant system overview as well as knowledge of components, which can only be obtained by combining information from many design documents into a 'mental' model. Only a few engineers might be capable of obtaining sufficient system overview such that they can determine and analyze (a part of) the actual system behavior. If this is the case, the system analysis results are based on personal capabilities, assumptions, and mental models, rather than on solid and objective experiments and proofs. In this chapter, we investigate how formal and executable models can be used to analyze the actual system behavior without a realized system.

In this chapter, we use the second type of the integrated approach as explained in the previous chapter to model and analyze multi-disciplinary system behavior. The expressivity of the χ language is used to represent the concurrent behavior of interacting processes, as well as important aspects of other paradigms than concurrent processes, e.g., various physical phenomena such as motion, pressure, and flow. By combining the χ models of the components, an integrated system model is obtained, which can be analyzed using various techniques and tools.

This chapter is mainly based on [Braspenning et al. 2008], in which co-author Bortnik provided the χ to UPPAAL translation scheme, and is organized as follows. Section 3.1 describes model-based system analysis techniques in general, particularly focusing on using the model checker UPPAAL to verify χ models. Section 3.2 answers QUESTION 1.3 by showing how these model-based system analysis techniques were applied in the EUV case study. The conclusions of this chapter are given in Section 3.3.

3.1 Theory: model-based analysis techniques

In literature, analysis techniques are often categorized into validation and verification techniques. Validation concerns the question "are we creating the right product?", usually involving user requirements, while verification concerns the question "are we creating the product right?", usually involving design requirements [Boehm 1981]. Although these simple definitions are widely accepted, the difference is not always clear and their usage is subjective; both terms are often used by different people to refer to the same type of analysis technique. In this thesis, this subjectivity is avoided by disregarding a strict categorization of analysis techniques, because the exact difference between the categories (validation or verification) is irrelevant in practice. In our view, the purpose of all analysis techniques is the same: to obtain system overview and to detect problems as early as possible in the system development process, no matter whether these problems are related to user or design requirements.

Considering the MBI&T method instantiation used in this thesis, we focus on model-based analysis techniques that can be used in the concurrent processes paradigm, particularly those that are suitable for process algebras such as χ. Common model-based techniques to analyze the behavior of concurrent processes are simulation and model checking, which were already introduced in the previous chapter. Simulation involves determining *some* behavior of a model, which is analyzed for its correctness by *comparing* it with the intended behavior, while model checking involves determining *all* behaviors of a model, which are analyzed for their correctness by *proving* that certain properties are valid. Typical properties that can be analyzed by model checking are deadlock freeness, i.e., there are no states in the behavior in which no further actions are possible, and livelock freeness, i.e., there are no states in the behavior in which actions are still possible, but no progression is made (e.g., looping behavior). Furthermore, model checking is often used to check safety properties, i.e., some 'bad thing' such as machine damage or a hazardous situation does not happen, liveness properties, i.e., some 'good thing' does happen (eventually), as well as system specific behavioral properties, involving, e.g., required action sequences or required system performance.

In the remainder of this chapter, we focus on the analysis of systems expressed in the χ language, using both simulation and model checking techniques. First, the timed discrete-event version of χ [Van Beek et al. 2005] as used in this thesis is explained informally. We refer to [Van Beek et al. 2005] and to [Van Beek et al. 2006a; Man and Schiffelers 2006] for a complete syntax definition and a formal semantics of timed χ and hybrid χ, respectively, and to [Systems Engineering Group 2007; Hofkamp and Rooda 2007] for more information on the implementation of the language and its tools.

The informal explanation of χ in the next paragraphs is inspired by [Trčka 2006]. The

atomic process terms of χ, explained below, are process constructors and cannot be split into smaller process terms. Note that, besides the explicit *delay* process term, some process terms can also implicitly delay. The *deadlock* process term δ cannot perform actions or delays. The *skip* process term performs an internal action τ and cannot delay. The *(multi)assignment* process term $\mathbf{x}_n := \mathbf{e}_n$, where $\mathbf{x}_n$ and $\mathbf{e}_n$ are vectors of n variables and expressions, respectively, assigns the value of the expression e_i to the variable x_i, $1 \leq i \leq n$. It does not have the possibility to delay. The *delay* process term Δe delays a number of time units equal to the value of the expression e. The *send* process term $h\,!!\,\mathbf{e}_n$ sends the values of the expression vector $\mathbf{e}_n$ along the channel h and cannot delay. The *delayable send* $h\,!\,\mathbf{e}_n$ behaves as $h\,!!\,\mathbf{e}_n$ but it can delay arbitrarily long. The *receive* process term $h\,??\,\mathbf{x}_n$ receives the values for the variable vector $\mathbf{x}_n$ over the channel h and cannot delay. The *delayable receive* $h\,?\,\mathbf{x}_n$ behaves as $h\,??\,\mathbf{x}_n$ but it can delay arbitrarily long.

χ contains several operators to create compound process terms, which are informally explained below. The *guarded* process term $b \rightarrow p$ behaves as p when the value of the boolean guard b is true and blocks otherwise. The *delay enabling* operator $[p]$ enables p to perform arbitrary delays, without changing the action behavior of p. The *sequential composition* $p; q$ behaves as p followed by q. The *alternative composition* $p \,[]\, q$ stands for a non-deterministic choice between p and q, which can only delay if both p and q can delay. The *repetition* operator $*p$ behaves as p repeated arbitrarily many times. The *guarded repetition* process term $b \overset{*}{\rightarrow} p$ is interpreted as 'while b do (skip; p)'. The *parallel composition* operator $p \parallel q$ executes p and q concurrently in an interleaved fashion. It can only delay if both p and q can delay. In addition, if one of the process terms can execute a send action and the other one can execute a receive action on the same channel, then they can also communicate. For example, the process term $h\,!\,1 \parallel h\,?\,x$ can execute the communication action on channel h and assign the sent value 1 to the receive variable x. The *scope* operator is used to declare local channels and variables, i.e., the process term $[\![\, \mathbf{h}_m, \mathbf{x}_n :: p \,]\!]$ behaves as p with local channels $\mathbf{h}_m$ and local variables $\mathbf{x}_n$. The *encapsulation* operator $\partial_A(p)$ disables all actions of p that occur in the parameter set A. The *abstraction* operator $\tau_I(p)$ 'hides' (renames to τ) all actions of p that occur in the parameter set I. The *urgent communication* operator $v_H(p)$ gives communication actions of p via channels from the parameter set H a higher priority than the passage of time, i.e., p cannot delay if a communication action is possible. Finally, *process definitions*, e.g., $\mathsf{proc}\ P(\mathbf{h}_m, \mathbf{x}_n) = [\![\, p \,]\!]$, can be used to define a parameterized process which can be instantiated many times, possibly with different parameters.

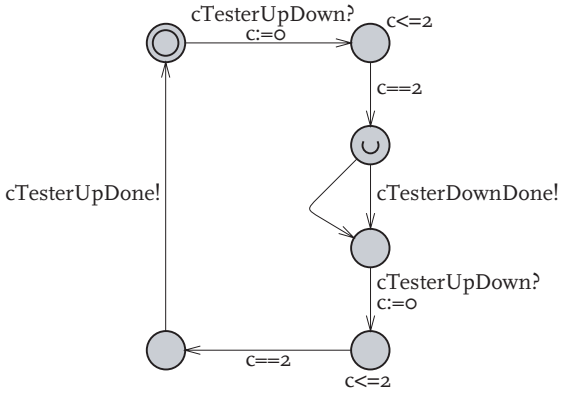
The χ language supports the following basic data types: booleans, naturals, integers, rational and real numbers, and typed channels. Most of the usual constants, operators and relations are defined for each data type and can be used together with variables to build expressions. Furthermore, χ provides mechanisms to build compound data types from all basic types except channels, e.g., tuples (notation $< type, type >$) and lists (notation $[type]$), and to define functions which enable functional programming.

With all these process terms, operators, data types, and functions, χ satisfies the model expressivity and compositionality requirements as described in Subsection 2.3.1. The expressivity and wide applicability of χ has been shown in several occasions, e.g., in the examples shown in [Man and Schiffelers 2006] and in industrial case studies as reported in, amongst others, [Haagh et al. 1998; Van de Mortel-Fronczak et al. 2001; Van Beek et al. 2002].

For example, Figure 3.1(a), taken from [Bortnik et al. 2005a], shows the χ model of the tester process of a turntable system that drills holes in products. The tester process is used to check whether the drilling process of the turntable system was successful, i.e., whether it drilled the hole completely through the product, see [Bortnik et al. 2005b] for more details. The tester process receives commands via the *cTesterUpDown* channel and uses an actuator to move the tester up or down. The tester process contains two position sensors, one above the product and one below the product, which send a signal via the *cTesterUpDone* and *cTesterDownDone* channels, respectively, when the presence of the tester is detected. The tester process has the following repetitive behavior. After receiving a command via *cTesterUpDown* to start testing, the tester moves down into the hole in the product, which is modeled by a delay of 2 time units (abstracting from motion details). If the tester reaches the lower sensor position, i.e., a signal is sent via *cTesterDownDone*, the hole was drilled deep enough and the test passes, otherwise the test fails and the product should be redrilled. In the χ model, these possible test outcomes are modeled by a non-deterministic choice between an undelayable *cTesterDownDone* !! (pass) and a skip process term (fail). After the test, the process waits for the command to move the tester up to the initial upper position ([*cTesterUpDown* ??]), which takes 2 time units. When the upper sensor position is reached, a signal is sent via *cTesterUpDone* and the test is finished.

```
proc Tester( chan cTesterUpDown ?
           , cTesterUpDone !
           , cTesterDownDone ! : void )=
[[ *( [cTesterUpDown ??]
    ; Δ2.0
    ; ( cTesterDownDone !! [] skip )
    ; [cTesterUpDown ??]
    ; Δ2.0
    ; [cTesterUpDone !!]
    )
]]
```

(a) χ model



(b) UPPAAL timed automaton

Figure 3.1: Example of χ to UPPAAL translation

To determine and analyze a possible behavior of a χ model, a simulator can be used. The basis of a simulator is a mechanism that is able to determine all possible actions for a certain state of a model and to execute such actions using an implementation of the model semantics. Based on this mechanism, called the 'stepper' for χ [Van Beek et al. 2006b], the principle of a simulator for χ processes is simple: determine all possible actions for the current state of the process, select one of them (non-deterministically), and execute this action to determine the new current state of the process. Simulation is continued either until the process and thus the simulation run is finished, or until no further actions are possible while the process is not yet finished, resulting in the deadlock process term δ.

As previously mentioned, simulation can only show that a model *might* have correct behavior, since not all possible behaviors are determined and analyzed. To analyze all possible behaviors of a χ model, we use UPPAAL [UPPAAL 2007], a tool for simulation and model checking of real-time systems that can be modeled as a network of timed automata [Bengtsson and Yi 2004]. UPPAAL has been used in a number of industrial case studies, see [UPPAAL 2007] for an overview. To be able to verify χ models in UPPAAL, a translation scheme from χ models to UPPAAL timed automata has been formally defined [Bortnik et al. 2005a] and the proofs of its correctness have been given [Bortnik et al. 2007]. Since it is not possible to define a general translation for all (combinations of) χ process terms, the translation scheme is defined for a subset of χ, which has been defined according to the following requirements:

- Make the translation as simple as possible. For this reason, a minimal subset of χ was translated. For instance, the delayable send process $h\,!\,\mathbf{e}_n$ is a syntactic extension, formally defined as $[h\,!!\,\mathbf{e}_n]$. Replacing $h\,!\,\mathbf{e}_n$ with $[h\,!!\,\mathbf{e}_n]$ does not change behavior of the model but makes it translatable.

- Make the translation as general as possible. For instance, a general translation of the guard operator $b \rightarrow p$ could not be defined, but there are many specific cases for which guarded process terms can be translated, namely guarded skip, guarded multi-assignment, guarded send and receive, and guarded repetition[1].

- Make the translated UPPAAL timed automata as readable as possible. For example, nested parallel composition can be translated as an alternative composition of all possible transitions [Man and Schiffelers 2006], but it would make the resulting automata less readable.

The translatable subset of χ contains all models of the form $\partial_A \upsilon_H(p)$, where A contains all individual send and receive actions, H contains all channels, and p does not contain abstraction, encapsulation, nor the urgent communication operator. Experience shows that most χ models of discrete-event systems are of this form. From now on, we only refer to the process p, omitting the top-level encapsulation and urgent communication operators. Since UPPAAL does not support nested automata, the process term p may either be a single sequential process or a parallel composition of multiple sequential processes, i.e., only one level of parallelism is allowed. The sequential processes of p may only contain the following translatable process terms and operators: skip, multi-assignment, delay, undelayable send and receive, guarded skip, guarded multi-assignment, guarded send and receive, guarded repetition, delay enabling operator, sequential and alternative composition, and repetition. We refer to [Bortnik et al. 2007] for more details on the χ to UPPAAL translation scheme.

As an example we consider the translation of the χ tester model from Figure 3.1(a), resulting in the UPPAAL timed automaton shown in Figure 3.1(b), taken from [Bortnik et al. 2005a]. The nodes depict the automaton locations (states) with invariants and the edges depict the transitions with guards, channel synchronizations, and actions. The upper left location is the

[1] In a more recent version of χ [Van Beek et al. 2008], the guard operator $b \rightarrow p$ has been replaced by a restricted set of guarded process terms: guarded skip, guarded multi-assignment, guarded send, and guarded receive.

initial location, depicted by the double circle. A clock variable c is used to model the delays in certain locations of the model. The incoming edge of a delay location contains a clock reset ($c := 0$), while the outgoing edge contains a guard on the duration of the delay ($c == 2$), which means that the delay location can only be left when two time units have elapsed. Together with an invariant in the delay location itself ($c <= 2$), which means that it is not possible to stay in the location for more than two time units, this models the $\Delta 2.0$ process term of the χ model. Furthermore, the non-deterministic choice between an undelayable send action and skip is modeled in the automaton by using multiple outgoing transitions and by making the location urgent, depicted by the U-sign in the location. An urgent location means that the automaton cannot delay in that location when an outgoing transition is enabled, which is always the case for the unlabeled transition that models the skip process term of the χ model.

The translation scheme from a subset of χ to UPPAAL timed automata has been implemented and integrated into the χ toolset. Together with the χ simulator, which is able to analyze χ models that are not restricted to the subset, the model checking capabilities of the χ toolset provide a powerful model-based analysis environment that satisfies the requirements for determining and analyzing model behavior, as described in Subsection 2.3.1.

The EUV case study, described in the next section, was the first industrial application in which the χ to UPPAAL translator was used. Therefore, one of the subgoals of this case study was to investigate the applicability and usability of the translator, e.g., whether the defined subset of χ that can be translated is sufficiently expressive in practice.

3.2 Practice: analysis of the EUV vacuum-source interface

As mentioned in Chapter 1, the EUV wafer scanner of Figure 1.1 is used as a case study throughout this thesis to answer QUESTION 1.3 on the practical applicability of the MBI&T method. In particular, the application of the MBI&T method to the EUV case study focuses on the interaction and time behavior of the vacuum system component C_v, which controls the vacuum conditions, and the source component C_s, which generates the EUV light. These components need close interaction to provide correct vacuum conditions and correct EUV light properties at all times. Since the internal states of these components are interdependent (e.g., the source may only be active under certain vacuum conditions to avoid machine damage), some combinations of component states are not allowed and should be prevented.

The EUV case study is a representative example of a high-tech multi-disciplinary system, as it includes multiple interacting components involving aspects from different disciplines, as schematically shown in Figure 3.2. The vacuum system component C_v contains a control part and a physical part. The C_v physical part consists of, for example, pumps, valves, and pressure sensors, arranged according to a particular architecture such that the pressure and flow in the EUV system can be controlled according to the system requirements. The C_v control part, consisting of electronics and (embedded) software, sends commands to the C_v physical part (e.g., 'turn on pump') and receives the status of the C_v physical part (e.g., 'pump is running') as well as the pressure as measured by the pressure sensors. The source component C_s also contains a control part and a physical part. The C_s physical part consists of a source of plasma-generated EUV light together with heaters, coolers, reflective optics,

and sensors to control the temperature and the light quality according to the system requirements. Similar to the C_v control part, the C_s control part uses electronics and (embedded) software to send commands to the C_s physical part (e.g., 'turn on source') and to receive the status (e.g., 'source is active') as well as the measured temperature.

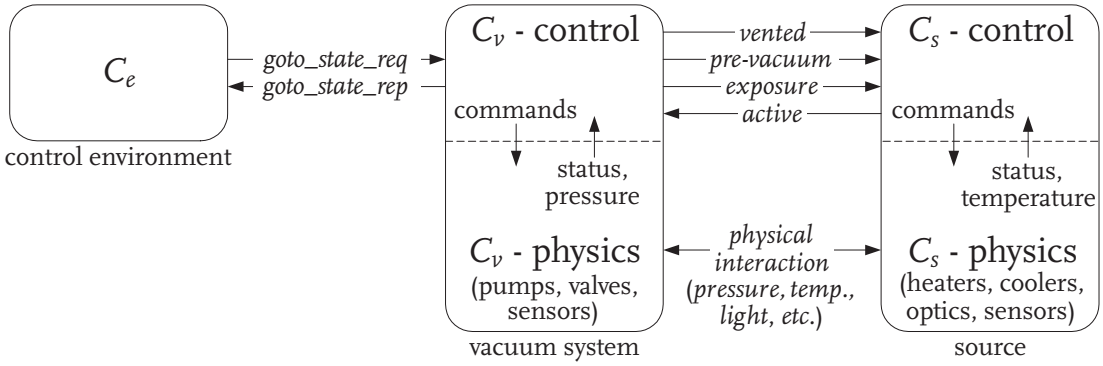


Figure 3.2: Components involved in the EUV case study and their interaction

The vacuum system and the source interact with each other on the physical as well as on the control level. On the physical level, for example, the pressure in the vacuum system influences the generation of EUV light, i.e., the vacuum conditions must be correct before the source is activated to prevent source damage. On the control level, C_v and C_s exchange information about their internal states via an interface consisting of four latches[2]: three latches from vacuum system to source and one latch from source to vacuum system:

Vented When active, this latch indicates that the vacuum system is vented.

Pre-vacuum When active, this latch indicates that the vacuum conditions are sufficient to activate the source, however not sufficient for exposure.

Exposure When active, this latch indicates that the vacuum conditions are correct for exposure.

Active When active, this latch indicates that the source is active and that the vacuum system is not allowed to go to the vented state (to avoid machine damage).

Besides these latches to interact with the C_s control part, the C_v control part provides a function *goto_state* to interact with the control environment of the system, which is represented here as component C_e. The control environment, e.g., a vacuum system operator or a higher level software controller, can send a request via *goto_state_req* to instruct the vacuum system to go to either the vacuum or the vented state. After receiving a request from C_e, the vacuum system sends back a reply 'OK' via *goto_state_rep*. Note that, by design, this reply

[2]Latch: electronic circuit based on sequential logic with inputs 'set' and 'reset' that is capable of storing one bit of information, i.e., a high or a low voltage. See Chapter 5 for more details.

does not indicate that the requested state is reached; it only indicates that the request is successfully received and that the vacuum system will perform the actions necessary to get to the requested state. The progress of these actions and the state of the vacuum system can only be observed via the vacuum system user interface without explicit notification that a certain state is reached, which was sufficient for the system design considered in the case study.

Throughout the thesis, the main focus of the case study is on the interaction and time behavior of the control parts of the vacuum system and the source, using abstractions of physical aspects such as pressure and temperature when these aspects are important for the analyzed system behavior. An example of such an abstraction is shown later in this section. The interaction behavior of the integrated system under nominal conditions is depicted by the message sequence chart in Figure 3.3. This figure shows the different states of the components and the communication between them for the vacuum sequence. In order to go to the requested vacuum state, first the 'vented' latch is deactivated after which the vacuum pumps are started and some initial preparation actions are executed by the source. After some pumping down, when the vacuum conditions are sufficient to activate the source, the 'pre-vacuum' and subsequently the 'active' latch are activated, and the source goes to the active state. Finally, when the vacuum conditions are correct for exposure, the 'exposure' latch is activated and the source goes to the exposure state. For the other way around, going from vacuum/exposure conditions to vented/inactive conditions, a similar, reversed sequence is specified.

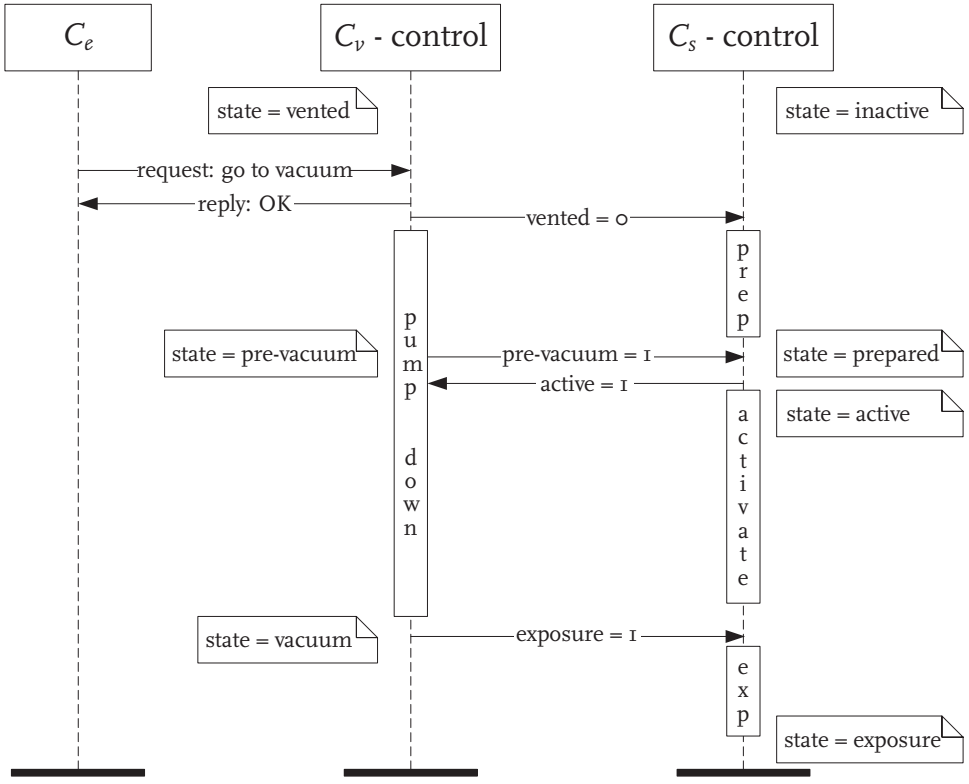


Figure 3.3: Nominal system behavior for the vacuum sequence

The nominal sequence described above does not cover preemption. The sequence can be interrupted at any time by a new request from C_e, and the vacuum system should handle these interrupts. For instance, when the vacuum system operator decides to go back to the vented state while the vacuum system is performing the vacuum sequence (i.e., going from vented to vacuum as shown in Figure 3.3), the vacuum system should immediately interrupt the vacuum sequence and start with the venting sequence to go to the vented state.

Finally, errors with different severity levels can be raised during operation, which lead to specified exceptional behavior that is not covered in Figure 3.3. For example, the source should leave the exposure state but remain active when a low level error occurs, however it should immediately deactivate and switch to manual mode when a high level error occurs. In manual mode, only a human operator is able to reactivate the source, after proper inspection of the error.

The following subsections describe how the MBI&T activities of the procedure described in Section 2.3 were applied to the EUV case study. First, the components of Figure 3.2, and the control parts in particular, were modeled as χ processes (MBI&T activity 1). Subsequently, the system model was analyzed using χ simulation and UPPAAL model checking (MBI&T activity 2). In these activities, the focus was on the order of actions during the vacuum and venting sequences, on the interrupt and error handling, on machine damage prevention, and on the time behavior of the system. Several requirements R concerning these issues will be model checked using UPPAAL.

3.2.1 Modeling the components in χ (activity 1)

Conforming to the first requirement as described in Subsection 2.3.1, the informal design documentation of the vacuum system and the source was used as the main information source for modeling. The system level design documentation (corresponding to D in Figure 2.8) described the interactions between the vacuum system and the source as shown in Figure 3.2. The design documentation for the source component, D_s, was reasonably complete and contained a good overview of the behavior in the form of a state diagram, which clearly showed the possible actions and communications for each state of the source. However, the design documentation of the vacuum system component, D_v, mainly focused on the physical actions for the vacuum and venting sequences, and did not contain information about the communication with the source on the control level. It became clear that the designers of the vacuum system were not yet fully aware of the communication behavior between their component and the source component. In general, the design documentation for both components mainly described the nominal behavior and hardly mentioned the handling of exceptional behavior, and also the action durations were not completely specified.

During our modeling activities, we obtained the missing information about the component designs by combining knowledge of both components and by discussion with the engineers involved. For example, the specifications of the communication of the vacuum system that was missing in D_v could be derived from the system and source design documents D and D_s. The updated and more complete design specifications were validated for their correctness by discussion with the designers of the vacuum system and source, corresponding to the model correctness requirement as described in Subsection 2.3.1. We

experienced that in most cases, the issues that arose during the modeling activities in fact indicated unknown or incomplete design issues such as missing states, obsolete states, or errors in the communication sequence. By incremental modeling, intermediate simulation, discussion, and design review, the specification documents were further corrected and completed, which also helped the engineers to obtain a better system overview. This is a practical example showing that the system development process, whether or not models are used, has a more incremental and iterative nature than the simplified 'sequential' process depicted in Figures 2.2 and 2.8.

There were some remaining modeling and design issues, for which no solution was available or for which the corresponding behavior was not known at all, even by the engineers involved. According to the engineers, the system behavior concerning these remaining issues was not important because it would never occur in the real system. This is a typical example of the current way of working in industrial practice that had to be taken for granted in the TANGRAM project. In order to deal with such incompleteness issues and statements that particular behavior would never occur, the behavior concerning these issues was abstracted in the model by using an explicit indication of 'undefined behavior'. In activity 2 of the case study, we verify whether this undefined behavior indeed can never occur.

The next paragraphs describe how the source C_s, the vacuum system C_v, and the control environment C_e were modeled as χ processes. Figure 3.4 shows how the χ processes of the system model (depicted by rounded rectangles) and the control level communication between them (depicted by arrows) are mapped onto the system design shown in Figure 3.2, abstracting from the physical parts and their physical interaction. The bold lines between processes of one component depict shared variables (two for the vacuum system and three for the source), which are used to exchange data between processes of one component.

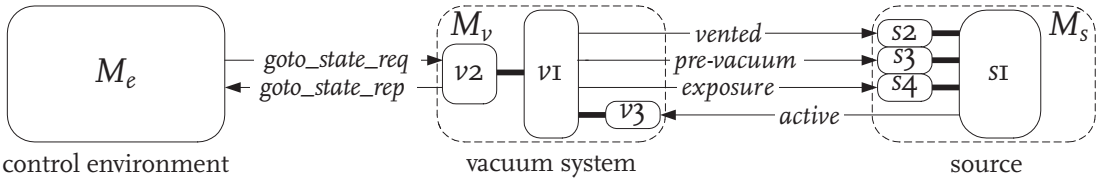


Figure 3.4: Process layout of the χ system model

The control environment is modeled as a single process M_e that can be configured to send requests and receive replies at certain points in time, depending on the analysis technique. For simulation and testing, specific scenarios with specific delays between the requests are used, while for verification, all scenarios (with any possible delay) are analyzed.

The vacuum system M_v is modeled as a parallel composition of processes $v1$ through $v3$, i.e., ($v1 \parallel v2 \parallel v3$). Process $v1$ is the core process and describes the internal behavior of the vacuum system, including the influence of relevant pressure levels, as explained in the following paragraphs. Processes $v2$ and $v3$ model the interaction with M_e and M_s, respectively. Besides receiving the incoming requests from M_e, process $v2$ also handles these requests, i.e., depending on the state of the core process $v1$, process $v2$ changes the variables shared with process $v1$ such that a new sequence is started or the current sequence is interrupted.

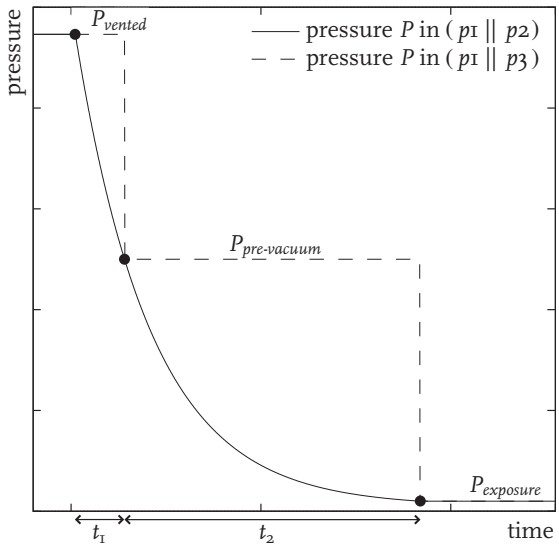
The internal behavior of the vacuum system as modeled in *v1* is influenced by the pressure level that changes due to actions in the physical part of the vacuum system, see Figure 3.2. In the χ model, this pressure is represented by a variable P. In principle, the physics behind pressure P can be modeled in the continuous-time paradigm using differential equations, which, combined with the discrete-event control part, results in a hybrid χ model. However, the simple pump example in Figure 3.5 shows that it is sufficient for the analyzed behavior in the EUV case study to use a discrete-event abstraction of the physics.

```
proc p1 =
  [[ pump_on := 1.0 ; vented ! false
  ; P ≤ P_pre-vacuum → pre-vacuum ! true
  ; P ≤ P_exposure   → exposure ! true
  ]]

proc p2 =
  [[ P' = pump_on * C_pump * ( P_end − P ) ]]

proc p3 =
  [[ pump_on = 1.0 → skip
  ; Δt1 ; P := P_pre-vacuum
  ; Δt2 ; P := P_exposure
  ]]
```

(a) χ model



(b) Pressure behavior

Figure 3.5: Modeling a simple pump: continuous-time vs. discrete-event

The left-hand side of Figure 3.5 shows three χ processes, $p1$ through $p3$, omitting the declaration and initialization of channels and variables for simplicity. Process $p1$ models a control process that performs a simple vacuum sequence, starting in the 'vented' state ($P = P_{vented}$) as in Figure 3.3. In this example, we assume that the vacuum sequence is not interrupted and that no errors occur. After turning on the pump and informing the source via the *vented* latch that the system is not vented anymore, process $p1$ waits until the pre-vacuum and the exposure pressure levels are reached and updates the latch values accordingly. Pump process $p2$ models the physics behind pressure P in the continuous-time paradigm, using a simple differential equation. In this equation, the pressure derivative P' depends on whether the pump is running (*pump_on*, controlled by $p1$), on the pump characteristics C_{pump} (here, we assume constant pump speed with no startup/brake effects), and on the difference between the end pressure P_{end} and the current pressure P. For the integrated system model ($p1$ || $p2$), the simulated trajectory of the pressure variable P is depicted by the solid line on the right-hand side of Figure 3.5. In the figure, the dots indicate the pressure levels P_{vented}, $P_{pre\text{-}vacuum}$, and $P_{exposure}$ at which the latch values are updated by $p1$. This example shows how the continuous-time physics of the vacuum system can be modeled in hybrid χ. We refer to [Man and Schiffelers 2006; Van Osch 2007] for more details about modeling in hybrid χ.

In the EUV case study in this thesis, however, only the latch interaction with the source is important for the analysis, i.e., the points in time at which the particular pressure levels are reached. Using the second type of the integrated approach as described in Subsection 2.4.1, we replace the continuous-time pump model in process $p2$ by the discrete-event pump model as shown in process $p3$. In this process, the pressure trajectories between the particular pressure levels are modeled by assigning new values for P at discrete points in time. The duration of the time delays between these discrete points in time, $t1$ and $t2$ in the example, can be calculated using a continuous-time pump model as in process $p2$. For the integrated system model ($p1 \parallel p3$), the simulated trajectory of the pressure variable P is depicted by the dashed line on the right-hand side of Figure 3.5. Independent of whether the continuous-time pump model in $p2$ or the discrete-event pump model in $p3$ is used, control process $p1$ updates the corresponding latch values at the same points in time. This means that the discrete-event abstraction of the physics behind pressure P does not influence the system behavior that is important for the analysis, i.e., the control level interaction and the time behavior of the integrated vacuum system and source. This example of multi-disciplinary modeling using the second type of the integrated approach as described in Subsection 2.4.1 shows that the expressivity of χ can be effectively used to model equivalents of other paradigms than the concurrent processes paradigm in an abstract way.

The source M_s is modeled as a parallel composition of processes $s1$ through $s4$, i.e., ($s1 \parallel s2 \parallel s3 \parallel s4$). Process $s1$ is the core process that models the internal behavior of the source, while processes $s2$, $s3$, and $s4$ model the latch interaction with the vacuum system. To give an impression of the χ model, a part of the source model M_s is shown in Figure 3.6, in which process $s1$ is shown on lines 2–19, and the processes $s2$, $s3$, and $s4$ are shown on lines 20, 21, and 22, respectively. For simplicity, the declaration and initialization of channels and variables are omitted and the model of the core process $s1$ only shows the behavior in the active state. All omitted parts are denoted by three dots (...).

The processes $s2$, $s3$, and $s4$ model the interaction with the vacuum system via the latch channels *vented*, *pre-vacuum*, and *exposure*, respectively, which is described in more detail in Chapter 5. The behavior of the repetitive processes $s2$, $s3$, and $s4$ is similar to each other. Each time when the vacuum system sends a signal via one of the latch channels, the received value is assigned to the corresponding boolean variable *vnt*, *pre*, or *exp*, respectively. These shared boolean variables indicate the states (active or inactive) of the latches, and can be read by process $s1$. When a new latch value is received, the auxiliary variable *newvalue* is set to true to notify process $s1$ that the state of some latch was changed.

The core process $s1$ models the internal behavior of the source as specified in the informal state diagram that was found in the source design documentation D_s. The state of process $s1$ is modeled by the variable *src_state*, which can take on the values *inactive*, *prepared*, *active* or *exposure*, see Figure 3.3. The variable *error* is used to model an error; *error* = 0 means that there is no error, the values 1, 2, 3, 4, 5 indicate different error levels with increasing severity. Finally, the variable *manual* indicates that the source has switched to manual mode due to a severe error, and the variable *undefined* indicates that the source has reached a state in which its behavior is undefined. Remember that undefined behavior was not described in the design documentation and should never occur according to the engineers.

The behavior of the core process $s1$ is state dependent. This example only shows the

```
1   proc M_s =
2     [[ * ( src_state = inactive  → ...
3          [] src_state = prepared → ...
4          [] src_state = active   → skip
5          ;  ( error = 5 → skip; Δ60; manual := true
6             [] error = 4 → manual := true
7             [] error < 4 → skip
8              ; newvalue → newvalue := false
9              ; (   vnt                          → error := 5
10               []  ¬vnt  ∧  pre   ∧  exp        → src_state := exposure
11               []  ¬vnt  ∧  ¬pre  ∧  exp        → undefined := true
12               []  ¬vnt  ∧  pre   ∧  ¬exp       → skip
13               []  ¬vnt  ∧  ¬pre  ∧  ¬exp       → Δ60
14                                                  ; src_state := prepared
15                                                  ; active !! false
16             )
17           )
18         [] src_state = exposure → ...
19         )
20     || * ( vented ?? vnt; newvalue := true )
21     || * ( pre_vacuum ?? pre; newvalue := true )
22     || * ( exposure ?? exp; newvalue := true )
23     ]]
```

Figure 3.6: Part of the source model M_s in χ

behavior in the active state, in which the process checks if there is an error and, depending on the severity of the error, takes certain actions to prevent further problems. For example, if *error* = 5 (line 5), the source deactivates the EUV light source, modeled as a delay for 60 time units, and switches to manual mode. Again, this is an example of multi-disciplinary modeling using the second type of the integrated approach as described in Subsection 2.4.1, i.e., the details of deactivating the EUV light (which might involve complex physical behavior) source are abstracted and replaced by a time delay.

If there is no severe error (*error* < 4, line 7), the source checks if any of the latch values has changed, indicated by the variable *newvalue*. If the *newvalue* guard is false, the process waits until it becomes true, i.e., until a new value is received via one of the latches. If *newvalue* is true, it is reset to false without a delay and the process continues by checking the current values of the three incoming latches (*vnt*, *pre*, *exp*) to determine the actions that should be performed (lines 9–13). If *vnt* = true (vacuum system is vented), the variable *error* is set to the highest error level 5, since the source is still in its active state. Otherwise, if *pre* and *exp* are true, the state of the source is switched to the exposure state, corresponding to the nominal behavior of the vacuum sequence. The situation in which *pre* is false and *exp* is true is one of the situations that should never occur and for which the behavior is undefined (*undefined* := true). If *pre* is true and *exp* is false, no action is taken (skip) and the process

remains in the active state. If both *pre* and *exp* are false, the process performs the actions required to go back to the prepared state, which are modeled by a delay of 60 time units. After switching to the prepared state, the 'active' latch channel to *v*3 of the vacuum system is deactivated (*active* !! false).

In the model shown in Figure 3.6, the process term skip, i.e., an internal action that cannot delay or change variables, was used for different reasons. In lines 4 and 7, we used skip to avoid guarded alternative composition ($b \rightarrow (p1 \,[]\, p2)$), and nested guards ($b_1 \rightarrow b_2 \rightarrow p$), which had not been implemented yet in the χ toolset. Then, in line 5, skip was included to make the choice immediate and to avoid that the required delay of 60 time units is interrupted when, in the meantime, the error level is lowered and another alternative is selected, which is undesired behavior. Finally, in line 12, no latch value has changed which means that the source remains in its active state. By using skip, the process *s*1 returns to line 2 without delay and without changing variables.

In this chapter, the focus is on functional analysis of the integrated system behavior, for which the exact details of the component interaction, e.g., how the control environment sends a request to the vacuum system, are unimportant. Here, only the result of the interaction is important for the analysis, e.g., the fact that a request is received by the vacuum system at a certain point in time. This means that details of the component interaction can be abstracted from in the model, resulting in an abstract infrastructure model I_M, using only the parallel composition operator $\|$ to synchronize the component models on communication and time. With this abstract infrastructure model I_M, the system model $\{M_e, M_v, M_s\}_{I_M}$ is modeled in χ as the parallel composition of the component processes: $(M_e \,\|\, M_v \,\|\, M_s)$. Chapter 5 provides more details about modeling, analysis, and implementation of the component interaction in the EUV case study to enable model-based integration and system testing.

3.2.2 Integrated system analysis using χ simulation (activity 2)

In order to analyze the behavior of the integrated vacuum system-source model by means of simulation, different scenarios are needed that focus on different aspects of the system. A good information source for possible scenarios is the intended system behavior specified in the system requirements and design documentation, i.e., *R* and *D*. Unfortunately, only one scenario could directly be derived from the documentation used in the EUV case study. This scenario corresponds to the nominal behavior of the system.

From ASML testing experience, it is known that analyzing only the nominal behavior is not sufficient. In many cases, it is the exceptional behavior that gives the problems, since this behavior is less documented and thus less clear when compared to the nominal behavior. Therefore, it is important to analyze the exceptional behavior as soon as possible. Unfortunately, no simulation scenarios for exceptional behavior could be derived directly from the documentation. However, based on the system overview obtained by modeling the components, and by discussion with the involved engineers, four additional scenarios for exceptional behavior analysis were derived. These scenarios cover the behavior of the system when the vacuum and venting sequences are interrupted at certain points in time.

Besides incomplete documentation, there is another problem with the analysis of exceptional behavior in the current, non model-based, way of working. Since only realizations can

be used for system analysis, it may be difficult, expensive, or risky to create the non-nominal conditions needed to analyze the exceptional behavior, e.g., a broken component. In the MBI&T method, creating these non-nominal conditions using models is easier, less expensive, and less risky. For example, the model can easily be adapted such that a communication signal is send earlier or later than expected, or that a different signal or no signal at all is sent.

In the case study, we used specific configurations of the control environment model M_e to analyze the integrated system behavior for the five scenarios mentioned above, one with nominal and four with exceptional behavior. The simulation results were visualized by means of animated automata, message sequence charts, and Gantt charts (see Figure 3.7 for an impression), which greatly improved the interpretation and analysis of the simulation results.

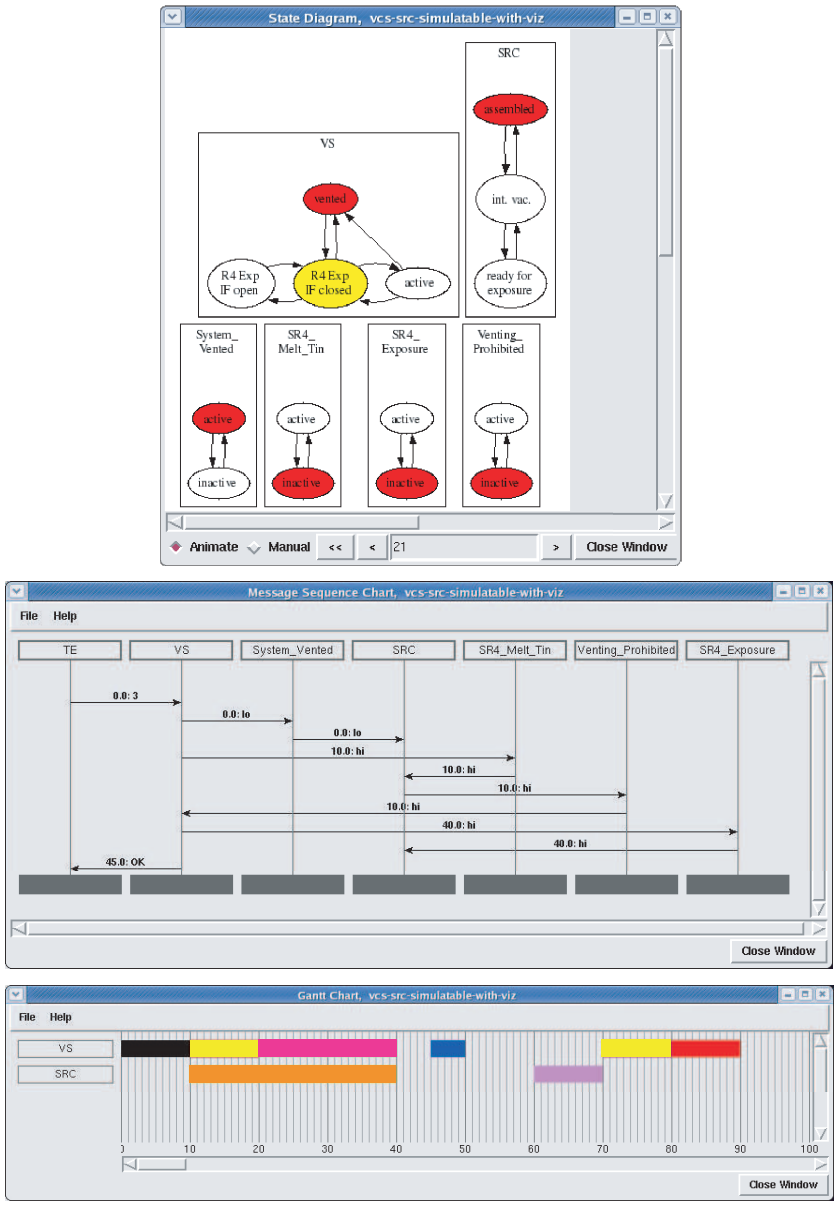


Figure 3.7: Visualization of simulation results: animated automata (top), message sequence chart (middle), and Gantt chart (bottom)

One situation with incorrect behavior was detected, which surprisingly also occurred in the nominal behavior scenario. The incorrect behavior occurs during the venting sequence, in which the vacuum system first deactivates the 'exposure' and 'pre-vacuum' latches. According to its design, the source should first observe the deactivated 'exposure' latch and perform some actions before observing the deactivated 'pre-vacuum' latch. However, the vacuum system was designed to deactivate both latches at the same time, which means that the source can also receive the deactivated 'pre-vacuum' latch during the actions it performs to reach the prepared state, or even before receiving the deactivated 'exposure' latch. In both cases, the source raises an error and switches to manual mode, which is certainly not acceptable for nominal behavior. Further diagnosis showed that this incorrect behavior indeed was an integration problem between the vacuum system and the source, which could now be solved early in the design phase.

3.2.3 Integrated system analysis using UPPAAL (activity 2)

During simulation, some problems were discovered and solved, which increased the confidence, but did not prove the correctness of the model. To check whether the model behaves correctly in all possible scenarios and to gain more system overview, it has to be verified. To enable verification of all possible scenarios, the control environment model M_e was configured such that any possible delay can occur between subsequent vacuum system requests and the χ system model was translated to a network of UPPAAL timed automata using the automatic χ to UPPAAL translator as described in Section 3.1. Subsequently, the UPPAAL model checker was used to analyze particular properties of the integrated system model.

Since the original χ model was created without considering its translation to UPPAAL, it uses some process terms for which no translation is defined, e.g., the scope operator, process definition and instantiation, delayable send and receive, nested parallelism, and guarded delay. These process terms need to be transformed manually to make the χ model translatable. To remove the scope operators and the process definitions from the χ model, all variables were lifted to the global scope and given unique names such that the local scope operators and the process definitions can be removed without changing the behavior. As previously mentioned, the delayable send and receive process terms can be replaced by their formal definitions without changing the behavior. The original χ model contained nested parallelism of the form $p_1;\ ((p_2;\ p_3) \parallel p_4);\ p_5$. Process p_2 changes the value of a variable after which process p_4 should immediately perform an undelayable action and terminate. Closer analysis showed that this particular case can also be modeled as $p_1;\ p_2;\ p_4;\ p_3;\ p_5$ without using the nested parallel operator. Finally, the original χ model also contained guarded delays of the form $(b \rightarrow p)\ [\!]\ (\neg b \rightarrow \Delta d)$, where $p \in \{\text{skip},\ \mathbf{x}_n := \mathbf{e}_n\}$. In a way similar to the proofs of the χ properties in [Van Beek et al. 2006a; Man and Schiffelers 2006], it can be shown that this process term is bisimilar to the translatable process term $(b \rightarrow p)\ [\!]\ \Delta d$.

This part of the case study showed that the translatable subset of χ should be extended with the scope operator and process definition. Furthermore, the support for data types should be extended to avoid unnecessary model transformations such as the encoding of boolean variables and strings into integer variables, which was still needed in the case study to make the model translatable.

After transforming the original χ model as described above to make it translatable, it was translated automatically to eight UPPAAL timed automata including five clocks and 29 variables. Figure 3.8 shows the four UPPAAL timed automata generated for the partial χ source model M_s of Figure 3.6, where the same parts are omitted and denoted by three dots (...). The initial location S43 of *s1* relates to the alternative composition of different source states (starting on line 2 in Figure 3.6), where the source states are now numbered from 1 through 4. Location S38 relates to the alternative composition of different error levels (starting on line 5 in Figure 3.6), and location S36 relates to the alternative composition on different latch values (starting on line 9 in Figure 3.6).

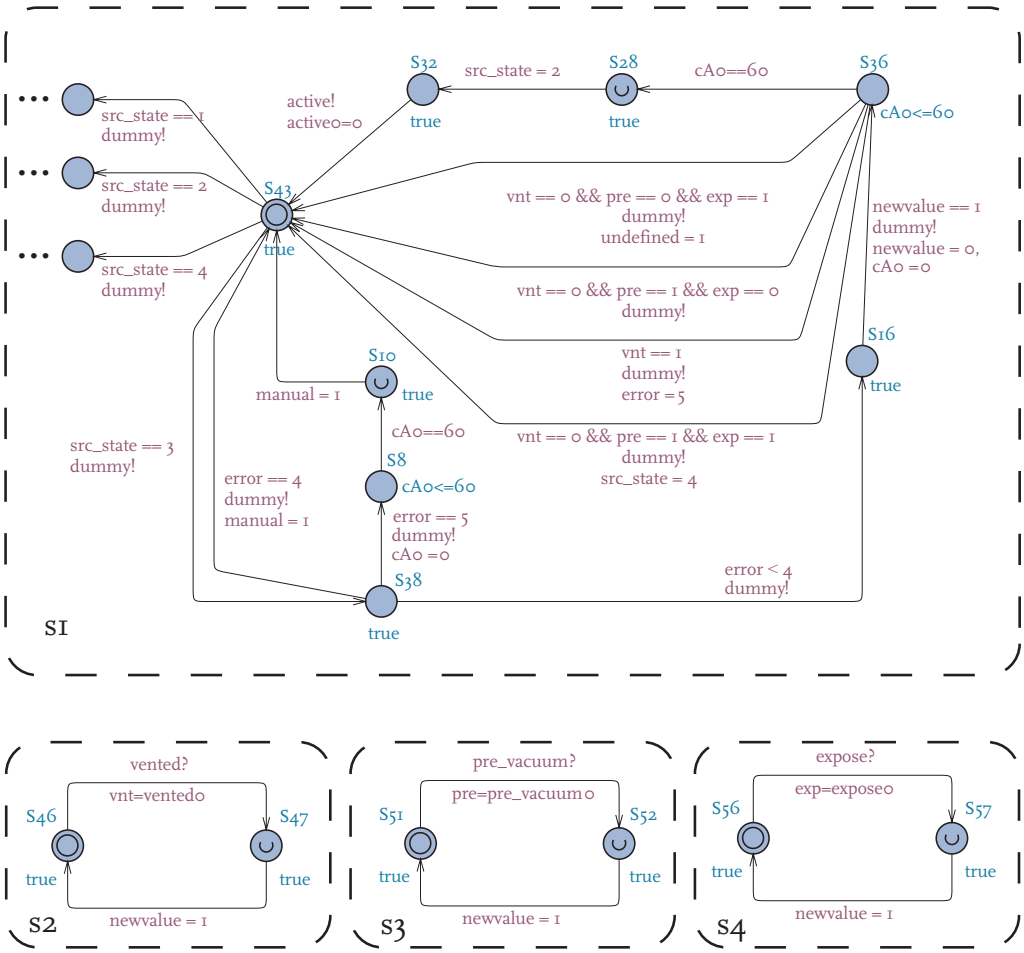


Figure 3.8: Generated UPPAAL timed automata for the source processes

The following properties, derived from the system requirements *R* and the system design *D*, were verified using the UPPAAL model checker. The properties are expressed as UPPAAL queries using timed computation tree logic (TCTL) [Bengtsson and Yi 2004], in which *A*[ ] *p* means that, for all traces, property *p* should hold *always*, while *A*<> *p* means that, for all traces, property *p* should hold *eventually*, i.e., after some time.

1. Deadlock freeness: *A*[ ] *deadlock imply env.end*, where *env.end* denotes the location of the control environment automaton indicating successful termination, which is the only location at which deadlock is allowed.
2. Livelock freeness: *A*<> *env.end*. When all requests from the control environment are processed by the vacuum system and the source, the control environment process terminates. This means that if the control environment automaton eventually reaches its end state *env.end* for all traces, there is no livelock in the system.
3. No undefined behavior: *A*[ ] *undefined* == 0. In some particular situations, the system behavior was not specified since it would never occur according to the engineers. These situations were modeled by assigning a non-zero value to the variable *undefined*, e.g., in line 11 of Figure 3.6.
4. No errors: *A*[ ] *error* == 0, where *error* is the variable indicating the error severity level of the source (*error* > 0), or the absence of an error (*error* = 0).
5. The vacuum system may not be vented while the source is active to avoid machine damage: *A*[ ] *not (vnt and act)*, where the variables *vnt* and *act* indicate the vented state of the vacuum system and the active state of the source, respectively.
6. The duration of the vacuum and venting sequences is at most 6 hours and 1 hour, respectively: *A*[ ] *vacuum imply clk* ≤ *21600* and *A*[ ] *venting imply clk* ≤ *3600*, where the variables *vacuum* and *venting* indicate which sequence is being performed, and *clk* is a clock variable used to determine the duration of the performed sequence (in seconds).

The translated model was verified in UPPAAL using the following options: generation of the fastest trace, breadth first search order, conservative space optimization, and state space representation using minimal constraint systems. The largest number of states (20510) was explored while verifying the first property. The total amount of memory used during model checking was just under 1 MB.

During verification of properties 1 and 2, two design errors were detected that caused deadlock. Both deadlocks were caused by non-determinism in the interleaving of the core process *v*1 and the interrupt handling process *v*2 of the vacuum system model. The way to handle this non-determinism in general was not specified in the design documentation, and model checking indicated this design incompleteness. One of the deadlocks occurred in the specific situation that an interrupt request was sent exactly at the start of the sequence. Again, this indicated an incompleteness in the design documentation, which did not mention the assumption that an interrupt could only occur after the start of a sequence. After informing the involved engineers about these design incompleteness issues, an alternative design for the interrupt handling process *v*2 was proposed. With this alternative design, and after restricting the control environment M_e such that interrupts can only occur after the start of a sequence, the model satisfies both properties 1 and 2.

Property 3 was satisfied by the model, indicating that the engineers were correct when they claimed that the situations with undefined behavior would never occur. Verification of property 4 detected the same design error as detected by simulation, i.e., incorrect behavior in the venting sequence, resulting in an error level larger than zero. Besides that, no other errors were raised in any trace of the model. Two minor mistakes were discovered

by verification of property 4 and 5, one modeling mistake and one mistake in the manual transformation from the original χ model to the translatable χ model.

To verify property 6, we decorated the model (using the approach from [Lindahl et al. 2001]) with additional boolean variables *vacuum* and *venting* to indicate which sequence is being performed and a clock *clk* to determine the duration of a sequence, without changing the behavior of the model. Both queries of property 6 were satisfied.

After discussing the detected design errors with the ASML engineers, fixes were applied to the design and, correspondingly, to the model. The fixed model was verified again and now all properties were satisfied by the model.

3.3 Conclusions

This chapter started with a description of techniques suitable for the analysis of system models in the concurrent process paradigm, i.e., simulation and model checking. These techniques are supported in the χ toolset, providing a χ model simulator and translators from χ to other formalisms that are suitable for model checking, e.g., UPPAAL timed automata.

The results of the EUV case study provide a positive answer to QUESTION 1.3: it is feasible and profitable to apply the MBI&T method in current industrial practice when no component realizations are available. The experience in the case study showed that the χ toolset, including the new χ to UPPAAL translator, is well suited for model-based system analysis, satisfying all requirements described in Subsection 2.3.1. The informal design documentation and discussions with involved engineers were effectively used as information source for modeling. The χ language is sufficiently expressive to model the considered aspects in the case study (supervisory machine control in software, interrupt behavior, electronic communication) and to deal with practical issues such as incomplete design specifications. The integrated χ system model could easily be obtained from the χ component models by using the available composition operators. The correctness of the χ models was evaluated by incremental modeling, intermediate simulation, and discussion with the involved engineers, demonstrating the incremental and iterative nature of a system development process in practice. The discussions with the engineers also showed that χ models are relatively easy to understand and that intermediate simulation results and visualizations are helpful to improve system overview. Finally, simulation and model checking were well suited to determine and analyze the system behavior before any component realization was available, which means that, for this MBI&T activity, the hypothesis on the applicability of the χ toolset is accepted.

Besides showing the practical applicability, the case study also showed relevant advantages for the system development process. First, the modeling activities helped to clarify, correct, and complete the design documentation, and to improve the system overview for the involved engineers. By simulation and model checking, a number of design and integration problems were detected and fixed during the system design phases. In terms of the T-Q-C business drivers, this means that the product quality Q could be improved earlier (lower time-to-market T) and with lower fix costs C when compared to current system development.

In total, five errors were detected by simulation and model checking. Three of these errors were design errors; only one of them was discovered by means of simulation. The

other two design errors, discovered by verification only, both concern the non-deterministic behavior of parallel processes. This behavior is difficult, if not impossible, to understand and to analyze by simulation or by reviewing the design documentation only. Nevertheless, when this non-deterministic behavior contains errors that remain undetected, it may cause 'strange' and unexpected problems on the system level, e.g., the system suddenly 'hangs' or it shows different behavior while the circumstances did not change. These problems are also difficult to diagnose, i.e., pinpointing the (unknown) errors in the non-deterministic behavior as the root cause for this strange and unexpected system behavior. This illustrates that verification should be used for designing real industrial systems, which often involve both high-level parallelism and non-deterministic behavior. Especially for systems with high risks for machine damage or human safety, analyzing all possible behaviors is essential in order to guarantee correct and safe system behavior under all circumstances. The other two errors were minor mistakes in the modeling and model transformation activities. To prevent model transformation errors, the translation scheme from χ to UPPAAL should be extended with the scope operator, process definition, and more data types. With these extensions, the translation scheme is well suited for translating most χ models. Unfortunately, the guarded delay and nested parallelism cannot be translated in the general case and these process terms should be avoided while modeling; otherwise they have to be transformed manually.

The fact that the largest state space generated for our system is reasonably small (order 20000 states) indicates that the approach is applicable to similar size or even more complex industrial systems. However, giving quantitative statements on the scalability of this approach is difficult. The UPPAAL model checker verifies the properties by generating the symbolic state space on-the-fly. This means that it does not need to generate the entire state space to verify a property, resulting in a significant reduction of the needed amount of time and memory. However, the size of the generated state space is, then, dependent on the property to be verified. Furthermore, the size of the state space heavily depends on the model itself, for instance, the number of automata, clocks, and variables, the level of non-determinism, and the structure of the automata. This is also recognized in literature, e.g., in [Diethers and Huhn 2004], which shows that a detailed but rather deterministic model could easily be handled, while the analysis of a small example with high non-determinism failed. The state space size is also influenced by model checker options such as depth-first or breadth-first search. This means that predicting the size of the state space, and thus making statements on the scalability of the approach, is difficult.

This chapter showed that model checking is a relevant analysis technique to detect problems in industrial systems and that the computational limits were not reached in the case study. Although the potential state space explosion problem remains the main obstacle when model checking more complex systems, the size of problems that can be handled increases due to sustained research work and tool improvements, e.g., a recently developed symmetry reduction technique [Behrmann et al. 2006] and efficient state space storage [David et al. 2003]. Besides the development of techniques to reduce the state space and memory usage, the computing power available for model checking also increases continuously, with faster processors, more memory, and distributed computing techniques [Behrmann et al. 2000]. All these techniques provide a wide range of model checking solutions to tackle problems of different types and sizes, see [UPPAAL 2007] for an overview.

CHAPTER 4

Component testing

How to test a component automatically?

In current industrial practice, a component realization is usually tested on the component level before it is integrated and tested with other components, i.e., on the system level. In many cases, the activities related to component testing are performed manually, e.g., the definition of tests based on documents that specify the requirements and design of the component. The MBI&T method uses models instead of documents to represent components, for example, to enable early analysis of the system behavior as described in Chapter 3. In this chapter, we investigate whether these models can also be used for automatic testing of component realizations when they become available.

Automatic testing is receiving lots of attention as an alternative to manual testing, which is often not sufficient anymore to detect all problems within the given amount of test time. A well-known and widely used technique to speed up component testing is to use automatic test execution techniques and tools [Fewster and Graham 1999]. However, these techniques and tools still require manual definition of the tests that need to be executed and the mechanism to determine the test outcome (pass or fail), which is often called the 'test oracle'. Test automation can be taken one step further when also the definition of tests and the test oracle is automated. In the model-based testing research field [Brinksma and Tretmans 2001], models are used as reference of correct behavior from which tests and test oracles can be generated automatically. Model-based component testing is a promising technique to automate testing, however in turn it depends on the availability, quality and completeness of models. In current industrial practice, models are not always available and their quality and completeness may not be sufficient for model-based testing. This is also recognized in [Willemse 2006], in which test automation is taken another step further by also generating the models for model-based testing.

In this chapter, we investigate whether the models developed in the MBI&T method have sufficient quality and completeness to use them for automatic model-based component testing. Since automatic testing of components focuses on reducing component risk rather than system risk, it does not directly answer one of the QUESTIONS in this thesis. However, in the context of the MBI&T method, automatic model-based component testing increases

the number of tests that can be executed per time unit. As described in Section 1.4, this increased risk reduction rate is a test phase improvement that contributes to the main objective of reducing the disadvantageous influence of the I&T phases on the T-Q-C business drivers.

This chapter is mainly based on [Braspenning et al. 2006b] and is organized as follows. Section 4.1 introduces the research field of model-based testing, particularly focusing on how the χ models of the MBI&T method can be used to automatically test component realizations. The applicability of model-based component testing in industrial practice is shown in Section 4.2, which is followed by the conclusions in Section 4.3.

4.1 Theory: model-based testing

In the model-based testing research field, theories, techniques and tools are developed that use models for automatic testing of component realizations. Test theories define a certain conformance relation between realizations and models, from which a test generation algorithm can be derived. Two important and preferred properties of generated tests are soundness and completeness with respect to the conformance relation. Soundness means that all tests generated by the algorithm will pass when the realization conforms to the model, while completeness means that for a realization that does not conform to the model, the algorithm can in principle generate a test that detects such a non-conformance. An implementation of the test generation algorithm, combined with functionality for automatic test execution, enables automatic on-the-fly testing of a component realization Z_i based on its model M_i, which satisfies the corresponding requirements for model-based component testing described in Subsection 2.3.2. In practice, such an on-the-fly test tool generates test inputs from the model M_i, provides these inputs to the realization Z_i, and checks whether the outputs observed from Z_i conform to the expected outputs as specified in M_i. This process is continued until either a non-conformance is detected, resulting in the test outcome 'fail', or until testing is stopped based on a certain criterion (e.g., test duration or test coverage), resulting in the test outcome 'pass'.

The main advantage of model-based testing is that tests are automatically generated and executed, which means that more tests can be executed per time unit. Furthermore, the soundness and completeness properties guarantee that the generated tests are reliable, i.e., they will not result in incorrect test outcomes and any non-conformances can in principle be detected. Note that this does not guarantee that all non-conformances *will* be detected. Finally, instead of maintaining large sets of tests, only the models used to generate these tests need to be maintained.

Model-based component testing is the topic of another part of the TANGRAM project [Tretmans 2007], in which the research is based on the input-output conformance (*ioco*) test theory [Tretmans 1996]. The *ioco* test theory defines a conformance relation between implementations expressed as input-output transition systems and specifications expressed as labeled transition systems. Informally said, an implementation i is *ioco*-correct with respect to its specification s if and only if all traces (with inputs, outputs, and quiescence which means no output) that can be executed by i based on inputs specified in s, can also be executed by s. The fact that quiescence can also be included in the tests improves the expressivity

and coverage of tests, e.g., it allows explicit testing of the *absence* of output between two consecutive inputs. The *ioco* test theory also defines a test generation algorithm that produces a sound and complete test set, which is implemented in the test tool TorX [TorX 2007].

In this chapter, the *ioco* test theory and the TorX test tool are used to investigate whether the models developed in the MBI&T method can be used for automatic testing of the component realizations. Although extensions towards timed testing [Bohnenkamp and Belinfante 2005], hybrid testing [Van Osch 2006] and testing with more complex data [Frantzen et al. 2006] were developed in the Tangram project, the work described in this chapter is based on a version of TorX that only supports model-based testing of untimed discrete-event systems without complex data.

In this thesis, the MBI&T method is instantiated with the χ process algebra language and the corresponding toolset. Although the semantics of χ models is defined in terms of transition systems [Van Beek et al. 2005, 2006a; Man and Schiffelers 2006], on which also the *ioco* test theory is based, χ is currently not supported by TorX. However, TorX does support Promela [De Vries and Tretmans 2000], the specification formalism for the model checker Spin [Spin 2007]. Promela and Spin can also be used to verify χ models [Bortnik et al. 2005b], using a translation scheme from χ to Promela as proposed in [Trčka 2006]. By combining the χ to Promela translation scheme and the Promela support of TorX, it is possible to automatically determine the conformance of a component realization with respect to the χ model. In the context of the MBI&T method, this approach for model-based component testing is visualized in Figure 4.1, which is based on Figure 2.6. In the figure, a component C_i is represented by a χ model $M_{i,\chi}$, which is subsequently translated into a Promela equivalent $M_{i,P}$. TorX is used to automatically generate tests from the model $M_{i,P}$ and to execute these tests on the realization Z_i, in order to determine whether Z_i is *ioco* conforming to $M_{i,P}$ and thus to $M_{i,\chi}$.

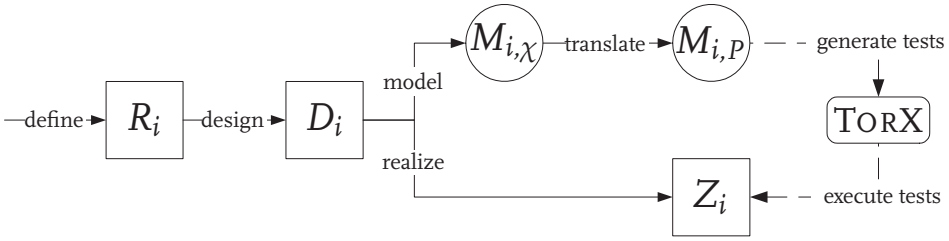


Figure 4.1: Model-based component testing with χ and TorX

Similar to the χ to Uppaal translation scheme used in Chapter 3, the χ to Promela translation scheme is also based on a subset of the χ language. Promela does not support nested parallelism, so a process term p is only translatable if it contains at most one level of parallelism, i.e., a single sequential process or a parallel composition of multiple sequential processes. The sequential processes of p may only contain the following translatable process terms and operators: skip, multi-assignment, delay, undelayable send and receive, delayable send and receive, guards, sequential and alternative compositions, repetition, guarded repetition, and the scope operator. Note that only simplified versions of guarded process terms

and the scope operator can be translated, for which some preprocessing steps have been defined to make any χ model translatable. Besides these process terms, the χ to PROMELA translation scheme also provides some support to translate compound data types such as tuples and bounded lists. We refer to [Trčka 2006] for more details on the χ to PROMELA translation scheme. Although time and data are supported by the χ to PROMELA translation scheme, these aspects are not supported by the version of TORX used in this chapter. This means that the subset of χ that is translatable to PROMELA should be sufficiently expressive for model-based testing with TORX.

When a χ model is translated to PROMELA, it has to be made suitable for model-based testing with TORX. In contrast to a PROMELA model used for simulation or model checking with SPIN, a PROMELA model used for model-based testing must be an *open* model, which means that some channels of the processes are not connected to other processes. These unconnected channels are the so-called *observable* channels, on which test inputs can be provided and test outputs can be observed [De Vries and Tretmans 2000]. In the PROMELA model, these observable channels are distinguished from the non-observable channels by a special channel attribute 'OBSERVABLE'. The TORX test generation algorithm only considers communication actions via these observable channels (test inputs and test outputs) and the absence of test outputs (quiescence). All other actions in the PROMELA model, including communication actions via non-observable channels, are considered to be internal τ actions in the generated tests. With the translation from χ to open PROMELA models, the TORX test generation algorithm can be used for model-based component testing, satisfying the test generation algorithm requirement as described in Subsection 2.3.2.

As visualized in Figure 4.1, TORX is connected to the model $M_{i,P}$ on one side (test generation), and to the realization Z_i on the other side (test execution). To connect TORX to a component realization, the abstract test inputs from the PROMELA model need to be transformed into real test inputs via the real interfaces of the component and vice versa for the test outputs. For example, this may involve calling certain functions of a software component realization and receiving the replies of these functions, including transformation or 'marshalling' of the function arguments and results. Transforming test inputs and test outputs and connecting to the real component interfaces is done by the adapter component of TORX, whose contents are case-specific and depend on the component interfaces that need to be connected and on the behavior that is tested. As shown in the next section, connecting to the real component interfaces, i.e., satisfying the corresponding requirement as described in Subsection 2.3.2, can be quite difficult in practice.

4.2 Practice: automatic testing of the laser source

This section describes how χ models were used for model-based component testing with TORX in an industrial case study. At the time of conducting the model-based component testing experiments in industrial practice, the realizations of the vacuum system and the EUV source, as analyzed in Chapter 3, were not available. Therefore, model-based component testing using χ models was applied to the light source of another type of wafer scanner, in which laser light is used to expose wafers. In particular, we focused on the interaction be-

tween a dose control component of the wafer scanner and the external laser source. Note that this is different from the vacuum-source interaction as analyzed in Chapter 3, since the wafer scanner type considered in this chapter does not have a vacuum system. The dose control component and the laser source need to interact in order to get the required 'dose' (amount of laser light) for each exposure. Experience has shown that the interaction between the dose control component and the laser is difficult to understand, integrate and diagnose. This is mainly caused by the fact that the laser source is produced by a third party manufacturer, meaning that the ASML engineers do not have full insight in and control over the behavior as implemented in the laser source. Therefore, correct integration of the dose control component and the laser source is an important aspect for the performance and reliability of the wafer scanner.

Due to safety and cost, a hardware laser simulator was used in the model-based component testing experiments instead of the real laser. This hardware laser simulator has the same software and electronic components and is specified to behave the same as the real laser, however it does not have the physical components to generate laser light. The laser simulator was developed by ASML and is used for testing the dose control software and electronics of the wafer scanner without the need for a real laser (including the required and expensive cleanroom facilities). In order to avoid faulty test outcomes or, even worse, faulty fixes in the dose control component when the laser simulator is used for testing, it is important that the behavior of the laser simulator satisfies the behavior specification of the real laser.

The remainder of this section describes how the MBI&T activities of the procedure described in Section 2.3 were applied to the dose control component and the laser source. First, the components were modeled as χ processes (MBI&T activity 1). The integrated χ system model was analyzed by χ simulation and, after translation to PROMELA, verified for certain properties using the SPIN model checker (MBI&T activity 2). Subsequently, the conformance of the hardware laser simulator with respect to the PROMELA equivalent of the laser source χ model was automatically tested using TORX (MBI&T activity 3a).

4.2.1 Modeling the components in χ (activity 1)

Similar to the modeling activities in Chapter 3, the requirements and design documents of the dose control component, the laser source, and their interaction were taken as main information sources for modeling the components. Again, the modeling activities helped in detecting and clarifying errors, inconsistencies, and incompleteness issues in the documents.

Figure 4.2 shows the process layout of the χ system model, in which the circles depict the processes and the arrows depict the channels that model the communication between processes. The dose control component is modeled as a single process *DC*, which can be configured (using an external configuration file) to execute specific command sequences, e.g., operational sequences as specified in the documentation. The laser source is modeled as the parallel composition of three processes: ($IO \parallel LC \parallel LS$). The I/O interface process *IO* receives commands from *DC* and, after a command has been handled by either *LC* or *LS*, it sends the responses back to *DC*. The laser communication process *LC* describes the

state behavior of the laser source. The process receives commands from *DC* that are passed through by *IO*. According to its behavior configuration (stored in an external configuration file), it performs the necessary actions (e.g., a state change) and creates the corresponding responses. Furthermore, *LC* contains the error handling of 'unknown' commands (unspecified commands) and 'bad context' commands (specified commands that are not allowed in a certain state). Finally, the laser state process *LS* keeps track of the laser state, which is used by *IO* to immediately respond to a laser state query command from *DC*, independent of whether the *LC* process is idle or busy.

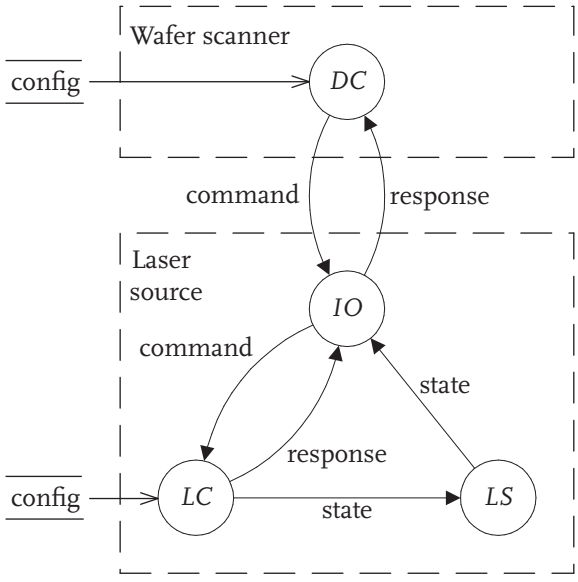


Figure 4.2: Processes and channels of dose control and the laser source

As previously mentioned, the version of TorX that was used in the case study did not support time and complex data. Therefore, time abstractions and data abstractions [Prenninger and Pretschner 2004] were applied to make the χ model suitable for model-based testing with TorX. These abstractions do not influence the state and communication behavior that is analyzed and tested in activities 2 and 3a of the case study, e.g., the ordering of events remains unchanged.

The resulting χ processes are *configurable* in a sense that the command sequences of the dose control component and the behavior of the laser source can be modified in external configuration files without modification and recompilation of the χ system model. This flexibility of modeling system behavior showed its advantage during the case study when it became clear that the hardware laser simulator was not available for a certain laser type and another laser type had to be modeled. The resulting χ system model contains 350 lines of code in total, including the necessary data definitions and functions.

4.2.2 Integrated system analysis using χ simulation and SPIN (activity 2)

The integrated system model developed in activity 1 of the case study was first analyzed using the simulator of the χ toolset. Several simulation runs were executed, in which the command sequences from the specification documents, e.g., for switching the laser source on and off, were specified in the configuration file of the dose control component process *DC*. Based on the simulation results, the laser source behavior conforms to the specification documents for all command sequences.

Subsequently, the χ system model was translated to PROMELA using the translation scheme from [Trčka 2006], allowing the verification of certain properties using the SPIN model checker. Since an automatic χ to PROMELA translator had not been implemented yet at the time of conducting the model-based component testing experiments, the translation was performed manually. This was a tedious and error-prone task, especially when changes were made to the χ model and, accordingly, to the translated PROMELA model. The resulting PROMELA system model contains 1850 lines of code, which illustrates the expressivity of the χ model, which contained 350 lines of code. Especially the number of lines needed to represent the rather simple data types (900 lines of PROMELA code) and the equivalents of the functions used in the χ model (300 lines of PROMELA code) drastically increases when translating χ to PROMELA, which is mainly caused by the laborious translation of compound data types such as tuples and lists.

Eight properties of the system were verified. The first property concerns the absence of deadlock or invalid end states, which is a standard model checking option in SPIN. The second property concerns a system invariant related to the precondition and the postcondition of the PROMELA translation of a specific χ statement. In linear temporal logic (LTL) [Holzmann 1997], this is expressed in the formula: [ ] (*precondition* –> <> *postcondition*). Here, [ ] *p* means that property *p* should hold *always*, <> *p* means that property *p* should hold *eventually*, and *p* –> *q* means *p implies q*, i.e., if *p* holds, *q* should also hold. The other six properties concern model specific behavior. Two of them concern the allowed order of state transitions. For example, from the 'off' state, the laser state may only become 'standby' without being 'on' in the meantime, which is expressed in the LTL formula [ ] (*state_off* –> ¬*state_on* *U* *state_standby*), in which *p U q* means *p until q*, i.e., *p* should hold until *q* holds. The other four model specific behavioral properties concern all possible actions and responses to each command. For example, when the laser receives the 'go_off' command while it is in the 'off' state or in the 'on' state, it remains in the current state and responds with 'not_allowed', or, when it receives the 'go_off' command while it is in the 'standby' state, it goes to the 'off' state and responds with 'state_off'. This is expressed in the LTL formula:

$$
\begin{aligned}
[\,]\,(\mathit{cmd\_go\_off} \text{ –> } \langle\rangle\ \ & ((\mathit{state\_off}\ \ U\ \ \mathit{rsp\_not\_allowed}) \vee \\
& (\mathit{state\_on}\ \ U\ \ \mathit{rsp\_not\_allowed}) \vee \\
& (\mathit{state\_standby}\ \ U\ \ (\mathit{state\_off} \wedge (\mathit{state\_off}\ \ U\ \ \mathit{rsp\_state\_off})))))
\end{aligned}
$$

All these properties were verified and found to be correct. The results of simulation and verification in this activity gave sufficient confidence that the laser source model is a good

representation of the requirements and design of the real laser source, i.e., a good basis for automatic model-based testing of the hardware laser simulator.

4.2.3 Model-based testing of the laser source using TorX (activity 3a)

In this activity, the laser source model developed in activity 1 and analyzed in activity 2 was used for automatic model-based testing of the hardware laser simulator using TorX.

Model-based testing requires an open Promela model with observable channels, which was obtained by removing the *DC* process from Figure 4.2 and by giving the unconnected command and response channels of process *IO* the special attribute 'OBSERVABLE'. The resulting model of only the laser source contains 1000 lines of code, including 300 lines for representing all data definitions and 300 lines for representing all functions.

In reality, the interaction between dose control and the laser is established by two bidirectional communication interfaces: a serial RS232 interface for commands and responses and a parallel interface for multiple status signals. Also the hardware laser simulator provides these interfaces to its environment, which are usually connected to the real dose control software and electronics for testing. Unfortunately, direct access of these interfaces from outside, which is required for model-based testing with TorX, was limited. While direct access functionality was provided for the common RS232 serial interface, this was not the case for the parallel interface, which uses an ASML specific communication protocol that is embedded in the electronics of the dose control component. This limitation in interface access from outside drastically reduces the laser source state space that can be reached and tested automatically. For example, the 'on' state, which was included during Spin model checking (see last LTL formula of the previous subsection), can only be reached by using the parallel interface. Nevertheless, the reduced behavior that can be tested with serial communication only is still sufficient to demonstrate automatic testing with TorX based on χ models. For correct communication over the serial interface, the adapter component of TorX is used to transform the abstract model commands into real laser commands (e.g., a left justified string of 128 bits) and to send these real commands over the serial interface using the provided direct access functionality. The adapter also receives the real responses from the laser source and transforms them back into the abstract responses as used in the Promela model. This experience shows that connecting the test tool to the realization, i.e., satisfying the corresponding requirement as described in Subsection 2.3.2, can be quite difficult in practice and that limited connectivity can drastically reduce the test coverage.

Now both the model and the realization are connected to TorX, the conformance of the hardware laser simulator with respect to the model can be tested. For all three model-based test runs that were performed, the test inputs were randomly selected from the possible commands in the Promela model. The selected commands are sent to the laser simulator using the adapter component and the provided direct access functionality, and subsequently the responses from the laser simulator are observed and compared to the expected responses as specified in the model.

The first model-based test run had a limited depth (less than 20 events) and took less than ten seconds until a discrepancy between realization and model was detected. To clarify this discrepancy, Figure 4.3 shows the state diagram of the laser source behavior that

was automatically tested using the serial interface only. In this figure, the nodes depict the states of the model, the solid edges depict the commands sent over the serial interface (starting with 'LS'), and the dashed edges depict the responses to the commands (starting with 'LS' or '??'). The central states at the top and bottom denote the actual laser states 'off' and 'standby', numbered '00' and '03', respectively. The 'state change', 'state query', and 'bad context' states are intermediate states between the commands and the corresponding responses. Note that any other command not shown in the figure results in an 'unknown command' response ('??=00').

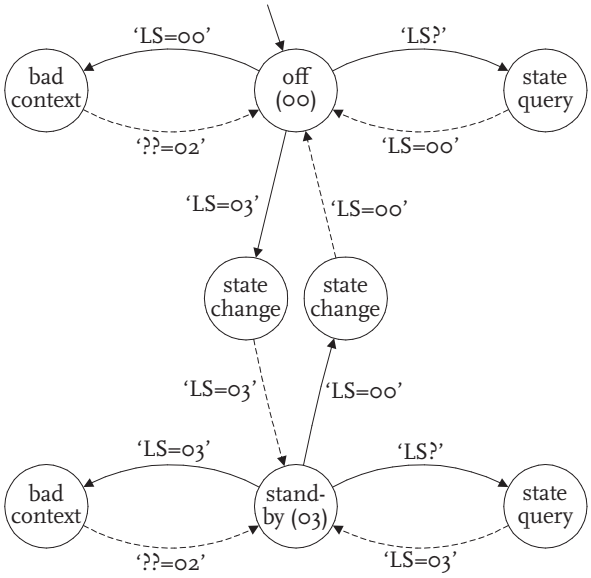


Figure 4.3: Laser source behavior that was tested

Figure 4.4 shows the message sequence chart of the first model-based test run. The figure shows the test inputs that TorX provides to the laser as well as the observed test outputs. After the fourth test input (laser state query command 'LS?'), the hardware laser source responds with 'LS=03', indicating that it is in the 'standby (03)' state. Subsequently, TorX provides the fifth test input 'LS=03' and observes the test output 'LS=03'. However, according to the behavior specification in Figure 4.3, giving the command 'LS=03' in the 'standby (03)' state, i.e., a command to go to the current state, should result in a 'bad context' response ('??=02'). This discrepancy between the observed test output and the expected test output results in the test outcome 'fail', as indicated by the last arrow in Figure 4.4.

The result of this model-based test run shows that the hardware laser simulator does not conform to the model and, correspondingly, to the laser source requirements and design. Further diagnosis showed that this non-conformance is due to an incorrect implementation of the error handling behavior in the hardware laser simulator. Directly fixing this error in the laser simulator software was impossible, because the required knowledge and tools were not available at that moment. Therefore, in order to enable further testing, a small modification was made in the model such that for the next test run there is no discrepancy between model and realization for the handling of the 'bad context' error.

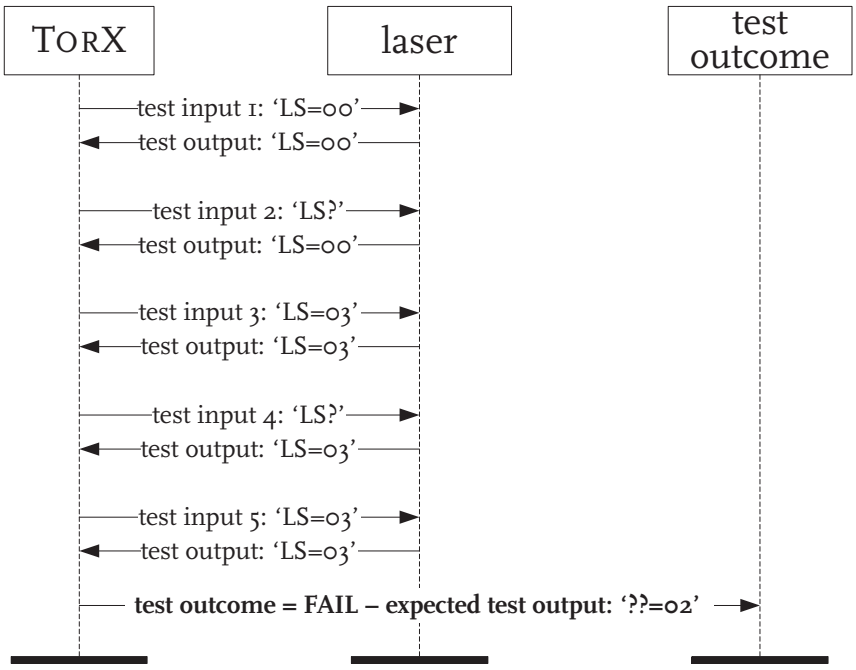


Figure 4.4: Message sequence chart showing the discrepancy

The second model-based test run had a limited depth as well (less than 20 events) and again took less than ten seconds until another discrepancy was detected, involving the handling of the 'unknown command' error. The laser source documentation, and thus the model as well, specified that any laser command other than 'LS=00', 'LS=03', or 'LS?' is unknown and should result in an 'unknown command' response (??=00). However, when such an 'unknown' command was selected during the test run, e.g., 'LS=01' which is an allowed test input, the laser simulator responded with the current laser state, as if the laser state query command 'LS?' was given. Further diagnosis showed that also this non-conformance is due to an incorrect implementation of the error handling behavior in the hardware laser simulator. To enable further testing, the set of allowed test inputs to be selected by TorX was restricted to known commands only, i.e., 'LS=00', 'LS=03', or 'LS?', such that for the next test run the 'unknown command' error handling behavior of the laser simulator would not be reached.

In the third model-based test run, the behavior related to the discrepancies detected in the first two test runs is not reached any more. This test run kept going for more than 15 minutes with a test depth of more than 1000 events, without detecting new discrepancies. Although the used version of TorX did not support coverage metrics, the test results of the third model-based test run and the size of the considered state space provided sufficient confidence that no other discrepancies would be detected.

The two implementation errors that were detected by automatic model-based testing are both related to the error handling behavior of the laser simulator. After discussion with the ASML engineers, it became clear that the laser simulator was mainly used for testing the dose control component under nominal behavior conditions. Although the errors may

appear to be trivial and should normally not be encountered during nominal testing with the laser simulator, this case study showed that such errors are not easily detected in the current way of working, and that a more systematic approach such as model-based testing certainly has potential. Furthermore, when these errors would remain undetected, they may still have a substantial impact in the case that the dose control component contains errors related to the implementation errors in the laser simulator. In that case, the errors in the dose control component may not be detected when the development tests rely on the laser simulator, which may cause problems later when the dose control component is used together with a real laser in a real production environment.

4.3 Conclusions

This chapter described how the models developed in the MBI&T method can be used to automatically test the component realizations, in particular using χ models and the test tool TorX. Although model-based component testing does not directly answer one of the Questions since it focuses on component risk rather than on system risk, it does increase the risk reduction rate, which is a test phase improvement as proposed by the second solution in Section 1.4. As such, it contributes to the main objective of reducing the disadvantageous influence of the I&T phases on the T-Q-C business drivers.

The proof of concept showed that χ models can be used for model-based testing with TorX, satisfying the requirements described in Subsection 2.3.2. Nevertheless, further enhancements are required to improve the general applicability in industrial practice. First, the expressivity of the χ language regarding state behavior, time behavior, and data types could not be fully exploited due to limitations of the (manual) χ to Promela translation process and the untimed discrete-event version of TorX used in the case study. This means that not all aspects of the system behavior, which usually include at least some time behavior, can be covered by model-based component testing and that other test activities are still required. This situation already improved since the χ to Promela translation scheme was recently implemented and integrated into the χ toolset, and new versions of TorX were developed that extend the expressivity of model-based testing with data [Frantzen et al. 2006], time [Bohnenkamp and Belinfante 2005], and hybrid behavior [Van Osch 2006]. Furthermore, TorX has recently been extended to support untimed discrete-event χ models as direct input formalism. The ability to use χ directly for model-based testing with TorX also suits the new versions of TorX, since their input formalisms are also closely related to χ. For example, the timed version of TorX is based on timed automata, the same class of models as used for Uppaal model checking in Chapter 3 that can also be expressed in timed χ [Van Beek et al. 2005], and a prototype tool for hybrid model-based testing is entirely based on hybrid χ [Van Osch 2007]. With these enhancements, the expressivity of χ models can effectively be used for automatic model-based testing, which means that, for this MBI&T activity, the hypothesis on the applicability of the χ toolset is accepted. However, as recognized in [Willemse 2006], obtaining models with sufficient quality and completeness for model-based testing remains difficult in current industrial practice.

The case study also showed that connecting the test tool to the realization under test can

be quite difficult and should be taken into account when deciding where to apply model-based component testing. Otherwise, as shown by the inability to use the parallel interface in the case study to provide test inputs, limited connectivity can drastically reduce the test coverage of model-based component testing. To improve this, the TorX adapter should support more generic connectivity techniques, rather than case-specific and ad hoc solutions. For example, the generic infrastructure to test ASML software components as developed in another part of the Tangram project [Denissen 2006; Tretmans 2007], as well as the infrastructure to connect models and realizations described in Chapter 5, could be used to improve the connectivity of the test tool. Unfortunately, these Tangram project results were not yet available or implemented at the time of conducting the industrial case study described in this chapter, leaving their application to model-based component testing as future work.

Besides showing the practical applicability, the case study also illustrated the profitability of model-based system analysis and model-based component testing using χ, Spin, and TorX. These MBI&T activities helped to detect and clarify errors, inconsistencies, and incompleteness issues in the documentation, and provided automatic detection of non-conformances between a component realization and its requirements. In terms of the T-Q-C business drivers, the case study did not really reduce the time-to-market T, since the tested laser source was already in operation at customer sites and no major problems were detected. Nevertheless, the case study provided a proof of concept showing that the test execution speed and thus the risk reduction rate can be increased, which potentially reduces time-to-market T in future applications. Furthermore, the detected non-conformances in the error handling behavior of the laser simulator did improve its quality Q, which potentially reduced the costs C in the case that related problems in the dose control component remain undetected when the laser simulator is used for testing.

In the model-based test runs that were performed, only the relation between individual commands and responses was tested using the regular testing features of TorX. However, it would also be interesting to test the specific behavioral properties as verified in activity 2 of the case study, involving relations of subsequent state transitions and combinations of responses and current states. Focussing model-based test runs towards such specific behaviors can be achieved by defining test purposes, a feature that is supported by TorX [De Vries and Tretmans 2001].

In principle, automatic model-based testing could be used for all test activities in the MBI&T method, including system testing in which the integrated system realization $\{Z_{1..n}\}_{I_Z}$ or a model-based integrated system with combined models and realizations is tested with respect to the integrated system model $\{M_{1..n}\}_{I_M}$. This is possible as long as the size and complexity of the integrated system model is not beyond the limitations of the test tool (model abstraction can be used to solve this issue), and as long as the test tool can access the required test interfaces of the integrated system realization. Moreover, research on compositional model-based testing [Van der Bijl et al. 2003; Benz 2007] could be used to imply the conformance of the integrated system based on the conformance of the individual components. However, a detailed investigation of applying model-based component testing techniques in industrial practice was not part of our research. Our goal in this chapter was to provide a proof of concept showing that the χ models developed in the MBI&T method can also be used for automatic testing of component realizations.

CHAPTER 5

Integration and system testing

How to go from models to realizations?

Although the previous chapters described how MBI&T activities 1, 2 and 3a support the analysis and testing of a system model (Chapter 3) or a component realization (Chapter 4), nothing is yet said about the final product of the system development process, i.e., the integrated system realization. This chapter concentrates on the remaining MBI&T activities 3b and 3c, in which models are used to test the integrated system at an early stage. In these activities, the interaction between models and realizations of components and the infrastructure *I* to establish this interaction play a major role.

In a high-tech multi-disciplinary system, many different interaction types are used, e.g., electrical signals between electronic components, function calls or message passing between software components, and memory mapped I/O between software and electronic components. Establishing the interaction between components requires that the infrastructure supports these different interaction types. When models are included in the I&T process, the (more abstract) interaction types used by the models should also be supported by the infrastructure. Furthermore, for model-based analysis of the integrated system behavior, the behavior of the infrastructure may also be important for the analysis, meaning that it should also be included in the model. Finally, the infrastructure should facilitate the replacement of a component model by the corresponding realization when it becomes available. This chapter investigates all these infrastructure requirements in more detail and describes how the infrastructure is used in the MBI&T method.

As explained in Section 2.4, combining models and realizations for early system analysis and testing is a well-known and common approach for some paradigms, e.g., hardware-in-the-loop testing and rapid prototyping in the dynamics and control paradigm, and hardware/software co-simulation in the real-time embedded software paradigm. However, it is less well-known and common for the concurrent processes paradigm, and this chapter contributes to this research area, with particular focus on the issues of synchronous and asynchronous communication. In this thesis, the components are modeled in a process algebraic language that uses synchronous communication, i.e., corresponding send and receive actions take place simultaneously. The reason for using a language with synchronous communication is that it allows easier reasoning about the interaction behavior, e.g., at which

points in time the interactions take place, because all processes involved need to be ready in order to execute the synchronous communication action. Furthermore, the fact that each process must be ready for interaction at specific points in time requires the engineer to think more carefully about the system, resulting in a better understanding of the system behavior. Finally, synchronous communication reduces the number of states in the model which improves the capabilities of model checking.

In contrast to the synchronous communication used in models, real systems often use asynchronous communication, i.e., send and receive actions do not take place simultaneously. This means that analysis results based on models using synchronous communication, e.g., correctness of behavioral properties derived from the system requirements, do not necessarily remain valid when the models are used for integration and testing with realizations in an asynchronous environment. Due to the different behavior of the asynchronous infrastructure, the models might interact differently with the other components, possibly resulting in wrong conclusions about the test results. Even worse, when certain safety requirements (regarding machine damage and human safety) are influenced by the infrastructure behavior, the safety, which was analyzed and found to be correct using the models, cannot be guaranteed in the realization environment, possibly resulting in hazardous situations. This example shows that it is important to ensure that the behavioral and infrastructural properties analyzed with a synchronous system model remain valid when models and realizations are integrated and tested using an asynchronous infrastructure. The approach described in this chapter shows how to deal with these infrastructural properties in the MBI&T method.

This chapter is mainly based on [Braspenning et al. 2007b,c] and is organized as follows. Section 5.1 contains some more background on the issues of synchronous and asynchronous communication. Furthermore, it introduces the different forms of infrastructure *I* and shows how the infrastructure is modeled, analyzed, and implemented within the MBI&T method. Subsequently, the applicability of this approach is shown by giving two examples of interaction types that are common for wafer scanners (Section 5.2) and by showing how the approach was used in the EUV case study to enable model-based integration and system testing (Section 5.3). Finally, the conclusions are drawn and discussed in Section 5.4.

5.1 Theory: infrastructure in the MBI&T method

Literature provides several approaches that deal with the implementation of synchronous models in an asynchronous environment. However, these approaches cannot be applied in the MBI&T method since the perspective on the goal of modeling is different. In most approaches found in literature, the goal is to automatically transform models into (software) realizations such that the communication remains synchronous. The models usually need some adaptations before they can be implemented in the asynchronous realization environment. For example, some approaches are based on a restricted subset of the modeling language or put certain constraints on the model [Groote 1988; Jones 2001; Mörk 2001]. This means that also the synchronization between components should be modeled and implemented within the corresponding subset and constraints, e.g., by adding explicit acknowledgement messages or by using semaphores. Other approaches augment the models with

additional communication messages according to some protocol in order to negotiate which components will communicate [Awerbuch 1985; Knabe 1993; Demaine 1998]. A common challenge in all these approaches is the correct implementation of the non-deterministic choice operator [Palamidessi 1997; Nestmann 2000], since this may offer many communication alternatives of which only one may be selected.

In contrast to these approaches, the MBI&T method focuses on detecting problems in the system *as it is designed* by the engineers. This means that the models are based on the 'as is' designs of the (software and hardware) components and the infrastructure. When the above mentioned approaches from literature would be applied, the models would need to be adapted for asynchronous implementation, e.g., language constructs that are not in the implementable subset would have to be removed or behavior for communication negotiation would have to be added. This means that the models would deviate from the 'as is' designs, which does not suit the MBI&T method. Using the 'as is' designs as basis for modeling also means that when a non-deterministic choice appears in a component design, it must also be included 'as is' in the model. The model can then be used to analyze potential problems caused by the non-deterministic choice in an asynchronous environment. In our view, solving the problems related to non-deterministic choice is not part of the modeling and asynchronous implementation activities (as in the approaches found in literature), but of the design activities performed by the engineers. Of course, when problems occur, they can be solved by applying the approaches found in literature to the design (and subsequently to the model).

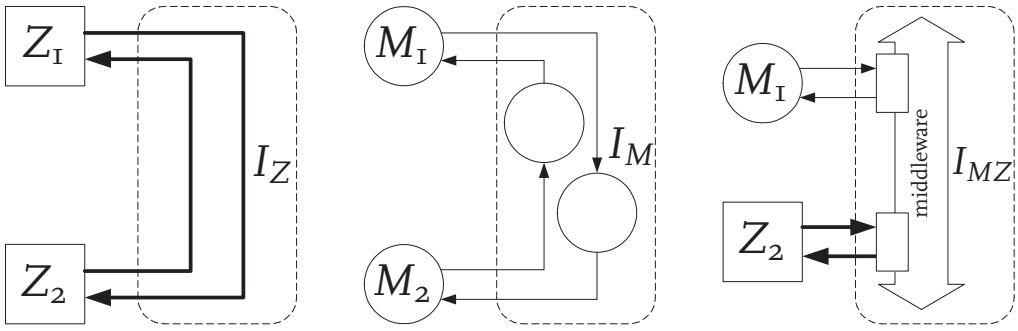
In the MBI&T method, the asynchronous component interaction as specified in the system design and realized in the infrastructure is explicitly expressed in the synchronous modeling language using additional processes. This is a long known approach [Milner 1989] that is often applied in process algebra [De Boer et al. 1992; Baeten and Bergstra 1992] and in synchronous programming, e.g., in the area of so-called globally asynchronous locally synchronous (GALS) systems [Halbwachs and Baghdadi 2002; Mousavi et al. 2004]. By modeling the asynchronous interaction behavior in a synchronous modeling language, we can use the powerful analysis techniques available for synchronous models to analyze the system behavior in an asynchronous environment. Most literature on this approach describes only how to model and analyze asynchronous system behavior using a synchronous language, while usually the implementation and execution of the resulting models in a real asynchronous environment is not discussed.

This chapter does cover the modeling, analysis, as well as the implementation of asynchronous component interaction in the context of the MBI&T method. First, the asynchronous interaction behavior of the components, as it is designed, is included in the synchronous system model for early system analysis. Subsequently, the modeled interaction behavior is implemented in an infrastructure that enables gradual replacement of models by realizations and early system testing of integrated models and realizations. This is done in such a way that the analysis results based on the synchronous system model remain valid during the integration and testing of models and realizations in an asynchronous environment.

Although we do not specifically focus on time behavior in this chapter, we use a similar 'model the behavior as it is designed' approach for time behavior as for the interaction be-

havior. Also in the area of time behavior, literature provides several approaches to transform models into realizations while preserving the proven time behavior properties [De Wulf et al. 2005; Huang et al. 2006]. In the MBI&T method, however, we model the time behavior according to the 'as is' component designs, possibly including solutions to prevent potential problems in the time behavior, e.g., a time synchronization protocol. Whether or not such solutions are included in the designs, we use the models, based on the 'as is' designs, for early system analysis and system testing to detect problems in the (time) behavior of the designed system as early as possible.

As described in Section 2.3, the MBI&T procedure consists of three main activities: modeling the components and their interaction, analysis of the resulting system model, and testing of integrated models and realizations of components. As shortly introduced in Chapter 2, three different forms of infrastructure are used in these activities: an infrastructure realization I_Z (also used in the current way of working), an infrastructure model I_M, and a model-based integration infrastructure I_{MZ}. As shown in Figure 5.1, these three forms imply different contents of the rounded rectangle that represented the 'generalized' infrastructure I in Figure 2.8. The remainder of this section describes each infrastructure form, particularly focusing on the issues of synchronous and asynchronous communication.



(a) I_Z (b) I_M (c) I_{MZ}

Figure 5.1: Different forms of infrastructure in the MBI&T method

5.1.1 Infrastructure realization I_Z

This is the 'real' infrastructure used in the system realization to implement the component interaction according to system design D, e.g., via signal cables and communication networks. For example, Figure 5.1(a) shows two component realizations Z_1 and Z_2 (boxes) and the infrastructure realization I_Z (bold arrows) that enables the communication between the components. For most interactions between electronic and software systems, on which we focus in this chapter, the communication in the real infrastructure is asynchronous, i.e., send and receive actions do not take place simultaneously.

5.1.2 Infrastructure model I_M

Besides that the components are modeled as M_i in the MBI&T method, also the infrastructure that enables component interaction is explicitly modeled and analyzed. The infrastructure can be modeled on different abstraction levels. During the initial modeling and analysis phases, there may be reasons to use synchronous communication in the model, i.e., completely ignoring the asynchronous behavior. One reason may be that a detailed infrastructure design is unavailable, but system model analysis with a synchronous abstraction of the infrastructure is still helpful. Another reason may be that the infrastructure details are not important for certain model-based analysis activities and would only increase the complexity and state space when they are included in the model. For example, analyzing the functionality of the system may be possible when only the result of an interaction is known (e.g., a message being transferred), without knowing exactly how that interaction is established. The latter reason was used in the EUV case study in Chapter 3.

Although the asynchronous infrastructure behavior may be ignored in the model initially as described above, it must be considered eventually. After all, the models developed in the MBI&T method will eventually be integrated and tested with realizations that do require an asynchronous infrastructure as in I_Z. When infrastructure details are considered during system modeling and analysis, the asynchronous behavior of the infrastructure as designed in D and realized in I_Z must be expressed in the modeling language that is used. For certain interaction types, the modeling language may have constructs to directly express that type of infrastructure. For interaction types that cannot directly be expressed in a modeling language, it may be possible to model their equivalent behavior. For example, asynchronous communication can be modeled by using additional processes in a synchronous language such as χ, which is a common and widely used approach [Milner 1989; De Boer et al. 1992; Baeten and Bergstra 1992; Halbwachs and Baghdadi 2002; Mousavi et al. 2004]. These additional processes are placed between two component processes to model the asynchronous behavior of that particular component interaction, e.g., a message buffer. Different types of component interaction may require different additional processes in the model, as shown for some examples in the next section of this chapter. We denote the modeling constructs used to express the component interaction behavior as the infrastructure model I_M. For example, Figure 5.1(b) shows four processes (circles) which, conforming to the process algebraic language, use synchronous communication (arrows). The processes M_1 and M_2 represent the component models and the unlabeled processes represent the infrastructure model I_M, resulting in asynchronous communication behavior between component models M_1 and M_2.

5.1.3 Model-based integration infrastructure I_{MZ}

To integrate combinations of models and realizations, a so-called model-based integration infrastructure I_{MZ} is needed that implements the component interaction as designed in D and modeled in I_M. Several requirements should be satisfied by I_{MZ}.

First of all, the communication paradigm of I_{MZ} should be asynchronous, since the realizations which are integrated by it will also communicate asynchronously. Furthermore, different types of component interaction may require different infrastructure behavior that

should be supported by I_{MZ}, similar to the different behavior that can be modeled in the infrastructure model I_M. Finally, I_{MZ} should allow easy integration of models and realizations, i.e., they can be connected via the infrastructure with minimal effort. To achieve this, the connection of components to the infrastructure should be independent of how other components are represented (model or realization) and independent of their exact name, location and interfaces. This makes the integration of components independent of whether models or realizations are used.

The last requirement, independency of connected components, is one of the main features of so-called *middleware*, which consists of intermediate software that connects software components with each other. Examples of middleware are remote procedure calls (synchronous and asynchronous), object request brokers, and message oriented middleware (e.g., publish-subscribe) [Hurwitz 1998]. The components only need to connect and communicate with the middleware, independent of the representation, name, location, and interfaces of the other components. As described in the next section, the MBI&T method instantiation in this thesis also uses a model-based integration infrastructure I_{MZ} that is based on middleware.

To connect the components to the middleware, the communication paradigms used by the component models or realizations must be adapted to the communication paradigm of the middleware. This is done by creating *connectors* for the models and realizations such that they communicate via the communication paradigm of the middleware. Different types of components, e.g., software components expressed in different languages, may require different connectors to be created. We denote the middleware together with the connectors for the models and realizations as the model-based integration infrastructure I_{MZ}. For example, Figure 5.1(c) shows the integration of a model M_1 and a realization Z_2 using middleware (vertical double headed arrow). Both components are connected to the middleware via connectors (small rectangles) that adapt the communication paradigm of M_1 (normal arrows, as in Figure 5.1(b)) and the communication paradigm of Z_2 (bold arrows, as in Figure 5.1(a)) to the middleware and vice versa. The middleware is configured such that the component interaction corresponds to that of Figure 5.1(a) and Figure 5.1(b), i.e., the middleware connects the outgoing communication of M_1 to the ingoing communication of Z_2 and vice versa.

In the MBI&T method instantiation used in this thesis, the following approach is used to model, analyze, and implement the infrastructure such that it supports all MBI&T activities while the analysis results based on the model remain valid during model-based integration and system testing. This approach is based on the MBI&T procedure of Section 2.3 and uses the different forms of infrastructure as introduced in this section. In MBI&T activities 1a and 1b, the components are modeled as $M_{1..n}$ based on component designs $D_{1..n}$ and the infrastructure is modeled as I_M based on system design D, respectively. Here, additional processes are used to model the asynchronous infrastructure behavior in a process algebraic language that uses synchronous communication. In MBI&T activity 2a, the component models $M_{1..n}$ are integrated using I_M, resulting in system model $\{M_{1..n}\}_{I_M}$. In MBI&T activity 2b, model-based techniques such as simulation and model checking are used to analyze the behavior of this system model, including particular properties related to the infrastructure behavior. Here, the entire system is expressed in a model, so determining and reasoning about the system behavior is completely based on the semantics of the modeling language,

including synchronization on communication and model time. In MBI&T activity 3, the infrastructure model I_M is implemented as model-based integration infrastructure I_{MZ}, resulting in $\{M_{1..n}\}_{I_{MZ}}$, in such a way that the infrastructure behavior of I_M is preserved (examples are given in the next sections). This model-based integration infrastructure I_{MZ} enables the replacement of a component model M_i with the corresponding realization Z_i when this realization becomes available (MBI&T activity 3b). The resulting model-based integrated system with combined models and realizations is used for early system testing in MBI&T activity 3c. Using real-time simulation techniques, the models are now executed as 'surrogate' realizations in an environment with asynchronous communication and real-time behavior. This means that determining and reasoning about the system behavior can no longer be based on the modeling language semantics as in simulation and model checking. Instead, the behavior of the model-based integrated system is determined by testing, using system tests that would also be executed when all component realizations are available and integrated.

In the following sections, we describe different applications of this approach, inspired by the ASML wafer scanners. Section 5.2 gives two intuitive examples of the approach applied to interaction types that are common for wafer scanners. Subsequently, Section 5.3 describes how the approach was used in the EUV case study.

5.2 Practice: common interaction type examples

As previously mentioned, the MBI&T method instantiation used in this thesis mainly focuses on the behavior of concurrent processes that communicate data. This behavior is an important aspect of software and electronic components and strongly relates to the interaction between these components. In general, concurrent behavior is less relevant for mechanical components, and these components themselves are often controlled via electronics and software. Therefore, we concentrate on software and electronic components and their interaction. In the following subsections, we give intuitive examples of the main software and electronic interaction types used in the ASML wafer scanner, namely function calls in software and sequential logic in electronics. For each interaction type, we explain the behavior and properties of the infrastructure realization I_Z and show how this behavior can be captured in a synchronous process algebra model I_M. The system model with all component models M_i and the infrastructure model I_M is used to analyze behavioral properties of the system and the infrastructure. Subsequently, we show how each interaction type can be implemented in a model-based integration infrastructure I_{MZ} using middleware.

To enable model-based integration and system testing in the instantiated MBI&T method, the χ toolset supports integration of χ component models with other (non-χ) components and real-time execution of χ models. This part of the χ toolset is based on a streaming XML technology called JABBER [JABBER Software Foundation 2007], extended with a protocol that enables communication via the publish-subscribe paradigm [Eugster et al. 2003; Millard et al. 2006]. The communication paradigm of publish-subscribe is simple. All publish-subscribe messages are related to a so-called 'topic', which identifies the contents and type of a message. Components can publish messages of a certain topic to the middleware, and components can subscribe to a certain topic, which means that they will receive all

published messages of that topic. The publish-subscribe paradigm satisfies all requirements for I_{MZ} as defined in the previous section. Communication via a publish-subscribe middleware is asynchronous since a message is first published to the middleware by a sending component, and then delivered by the middleware to the subscribed components, i.e., the middleware acts as a message buffer. Different types of component interaction, also modeled in different models I_M, can be configured by quality of service (QoS) properties of the publish-subscribe middleware, e.g., the number of messages to keep as history and the reliability of message delivery. Finally, since the components do not need to know the exact representation, name, location, and interfaces of the other components, the publish-subscribe communication paradigm is suitable to decouple components and thus to integrate both models and realizations with minimal effort.

With a publish-subscribe middleware, the transition from I_M to I_{MZ} is straightforward, namely all send and receive actions in the models must be related to corresponding write actions (to publish messages of a topic) and read actions (to receive messages of a subscribed topic) of the publish-subscribe middleware. Relating send and receive actions to corresponding write and read actions is accomplished in the model connectors, which can automatically be generated from the χ models using the χ toolset. The generated χ model connectors use the stepper [Van Beek et al. 2006b], also used for simulation, to determine the possible next actions of a model and to execute the selected actions in the model. Possible actions include internal behavior, communication behavior, and time behavior. Actions related to internal behavior, e.g., assignments, choices between alternatives, and calculations, are directly executed according to the χ semantics as implemented in the stepper. The execution of actions related to communication behavior, i.e., the send and receive actions, is implemented in the connector using the publish-subscribe middleware in the following way. A send action, e.g., $a!\,\mathrm{true}$, is executed by publishing the value (true) of the corresponding topic (a, the name of the channel) to the middleware, and by updating the state of the model to the state that follows the send action. In the case of a receive action, e.g., $b?x$, the connector determines whether a new value of the corresponding topic (b, the name of the channel) is available to be received by the model. Since the connector is subscribed to all topics that correspond to incoming channels of a component model, it will receive all published values of those topics, which are stored in a queue within the connector. Finally, the execution of actions related to time behavior, e.g., real-time delays and time-outs on receive actions, is implemented in the connector, using features from the event-driven networking framework TWISTED [TWISTED Matrix Labs 2007], in which also the publish-subscribe middleware is implemented. For time-outs on receive actions, the connector checks whether the read action corresponding to the receive action in the model can be executed within the specified amount of time. Otherwise, a time-out is triggered and the corresponding actions in the model are selected and executed.

The connectors for component realizations depend on the components themselves and may, for example, involve adapters that translate subscribed messages to function calls and function replies back to published messages, or software/hardware adapters that translate between software messages and electronic signals.

As shown in the remainder of this chapter, the combination of a publish-subscribe middleware with generated model connectors and case-specific realization connectors enables

model-based integration and system testing of χ models and realizations, satisfying the corresponding requirements described in Subsection 2.3.3.

5.2.1 Function calls (software)

A wafer scanner is controlled by a large amount of software, consisting of more than 12 million lines of code. The main interaction type used in this software system is the function call. A function call consists of an asynchronous request from a client to a server that provides the requested function, followed by waiting for an asynchronous reply from the server with the results of the function. The 'wait for reply' action can possibly contain a time-out that is triggered when the reply is not received within a specified amount of time. In practice, these time-outs are used to detect problems in the communication between client and server or in the function execution by the server.

There are two different types of function calls, blocking and non-blocking. In a blocking function call, no other statements may be executed between the request and the reply, while this is allowed in a non-blocking function call. Furthermore, blocking function calls do not use time-outs. Since a blocking function call is a special case of the non-blocking function call (with no statements between request and reply and no time-out), we only discuss the more generic non-blocking function call here.

Important properties of function calls as used at ASML are:

FIFO order Requests and replies between client and server may not overtake each other.

Buffer size The number of messages in the asynchronous communication buffer is limited.

Consistency The number of requests is equal to the number of replies or at most one larger (during function execution).

Wait/time-out A time-out may only be triggered when the reply buffer is empty for the specified amount of time since the start of the 'wait for reply' action.

Note that using the time-out as a detection mechanism for communication problems could be captured in a property 'time-outs may never occur', however this property is not related to infrastructure but to required system behavior.

Function calls with asynchronous communication can easily be modeled in a synchronous modeling language such as χ by including a buffer process between two communicating processes. The χ code of a buffer process B is shown in Figure 5.2, with two communication channels, input a and output b, for messages of type msg. The process repetitively checks for buffer overflow (lines 3–4), i.e., whether the length of message list xs exceeds the configured buffer size n. If this is not the case, the process continues with two alternatives (lines 6–7) of which the one that is enabled first will be selected. Either a new message x is received via channel a, which is then appended to xs (line 6), or, if xs is not empty, the head (first item) of xs is sent via channel b, after which the tail (all but first item) of xs remains (line 7).

Using multiple instantiations of buffer process B, we can model a function call as used at ASML as shown in Figure 5.3(a). For simplicity, the declaration and initialization of channels

```
1   proc B( chan a?, b! : msg, val n : nat )=
2   [[ var xs : [msg] = [], x : msg
3    :: * ( ( len(xs) > n   → overflow := true
4           [] len(xs) ≤ n  → skip
5           )
6         ; ( a?x           ;   xs := xs ++ [x]
7           [] len(xs) > 0  →   b!hd(xs); xs := tl(xs)
8           )
9         )
10  ]]
```

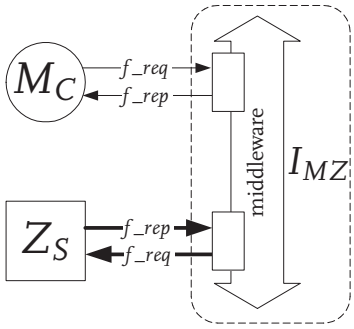
Figure 5.2: Buffer process B

and variables are omitted and only the body of the χ model is shown, namely four processes in parallel composition. The first process (lines 1–8) is a partial specification (denoted by ...) of a client that calls some function f. This function call is modeled as a sequential composition of sending an asynchronous request with the function arguments (*f_req*!*arg*, line 2) and receiving an asynchronous reply of the function with the results (*f_rep*?*res*, line 4). Between these two statements, other actions (denoted by ..., line 3) may be performed (not for blocking function calls). The possible time-out on the 'wait for reply' action is modeled as an alternative composition of the receive action (line 4) and a delay of t time units (line 5), which means that either the reply is received or the delay is finished, resulting in a time-out. Note that t is infinity (no time-out) for blocking function calls. The second process (lines 9–11) models the server, which repetitively waits for requests for the only function it provides, function f (more functions can be added in a similar way). Upon receiving a function call request from a client with certain arguments *arg* (line 9), the result of function f executed on *arg* is sent back as a reply (line 10). Finally, two instantiations of buffer process B (lines 12 and 13) are used to model the asynchronous communication. The buffer processes are connected to the request and reply channels of the client and server, similar to Figure 5.1(b). The buffer sizes are set to one since a client process may only call one function at a time. To simplify the example, we assumed that a function is required by only one client. More complex clients and servers with different properties can be modeled in a similar way, e.g., clients that make multiple subsequent calls, servers that allow multiple calls to be 'pending', or servers that provide functions required by multiple clients.

Using this infrastructure model I_M for function calls, we can include the infrastructural properties mentioned earlier in this section during system model analysis. In this chapter, we only make a reasonable case for the correctness of these properties by informal explanation, formal proofs are subject of further research. Due to the use of lists and their head and tail functionality in the buffer processes, it is not possible for two messages to overtake each other in the buffer, so the FIFO order property is satisfied. The validity of the buffer size property depends on the behavior of all components, and can be checked by performing a reachability analysis of the buffer overflow state of all buffer processes, e.g., by using a model

```
1   [[  ( ...
2       ; f_req ! arg
3       ; ...
4       ; ( f_rep ? res
5         [] Δt; time-out := true
6         )
7       ; ...
8       )
9   ||  *( f_req' ? arg
10        ; f_rep' ! f(arg)
11        )
12  ||  B( f_req, f_req', 1 )
13  ||  B( f_rep', f_rep, 1 )
14  ]]
```

(a) I_M



(b) I_{MZ}

Figure 5.3: I_M and I_{MZ} for function call example

checker as in Chapter 3. In the model of the server, each incoming request is immediately followed by sending the reply, so the number of requests is always equal to or at most one larger than the number of replies, satisfying the consistency property. For more complex server models, e.g., with functions required by multiple clients, request and reply counters can be added and a model checker can be used to determine whether the consistency property $0 \leq nr\_requests - nr\_replies \leq 1$ holds in all possible states. The wait/time-out property is covered in the infrastructure model I_M of Figure 5.3(a), because the communication in the χ model is urgent, meaning that a process may not delay if a communication action is enabled. This implies that the time-out Δt can only be triggered when after t time units the receive action *f_rep ? res* has not been enabled. Besides these already listed properties, two properties of blocking function calls, namely subsequent requests and replies (no intermediate statements) and no time-outs (t is infinity), can be checked by static analysis of the model structure, e.g., by a compiler.

When integrating models and realizations of components that use function calls to interact, the model-based integration infrastructure I_{MZ} can easily be implemented in the publish-subscribe middleware. Since the middleware uses asynchronous communication and acts as a message buffer itself, it is well suited as implementation of the buffer processes B from I_M in Figure 5.3(a) and of the real buffers used in the real function calls in I_Z. Figure 5.3(b) shows the implementation of I_{MZ} for the example of Figure 5.3(a), with a model of client M_C, a realization of server Z_S, and topics for the requests and replies. The client is configured as publisher of requests for function f (topic *f_req*), and it is subscribed to its replies (topic *f_rep*). The server is subscribed to function call requests for its provided function f, and publishes the corresponding replies.

As previously mentioned, the connectors are subscribed to all topics that correspond to the incoming channels of the component models, and all received values of these topics are stored in a queue within the connector. For the function call interaction type, the queue

behavior of the connectors is FIFO. As we can see in the next example, the queue behavior can be influenced via the QoS properties of the publish-subscribe middleware. With the FIFO queues in the connectors, a possible receive action $a?x$ in the model can only be selected if there is at least one message in the queue for the corresponding topic a. If the receive action is selected, the value of the oldest message in the queue is assigned to the receive variable x and the message is removed from the queue. In this way, the model-based integration infrastructure I_{MZ} behaves in a similar way as the infrastructure model I_M and the earlier defined properties, which were analyzed and found to be correct for I_M, remain valid in the realization environment.

For the integration of a client or server realization, the connector should translate between publish-subscribe messages and real function call requests and replies, including the transformation or 'marshalling' of the arguments. For example, when a server realization Z_S is integrated and when its connector receives a request of the topic *f_req*, the real function f of Z_S should be called by the connector, after which the result is published to the middleware with *f_rep* as topic. In the case that ASML software component realizations are integrated, the generic test infrastructure developed in another part of the TANGRAM project [Denissen 2006; Tretmans 2007] can be used to simplify the execution of real function calls by a middleware connector. Finally, when a client realization Z_C is integrated, its connector does not have to handle time-outs, because this is done by Z_C itself.

5.2.2 Sequential logic (electronics)

Many interaction types for electronic components are based on sequential logic, which depends not only on the current state, but also on the previous state. It is typically used to create memory in which values are stored as voltages in the circuits. Latches and flip-flops are well-known sequential circuits that appear in many forms for direct communication between electronic components (e.g., via signal cables) or for communication between software and electronics (e.g., via memory mapped I/O or distributed I/O). In all these forms of sequential logic, the sending component is able to set a certain value that is stored in the circuit, and the receiving component is able to observe or read this value. Taking the set/reset or SR-latch as a simple example, a sending component can set the SR-latch to active or reset it to inactive (i.e., high or low voltage). In most cases, the state of an SR-latch relates to some internal state of the sending component, e.g., 'standby', 'ready for next action', or 'error'. Via the SR-latch, the receiving components can observe this internal state at all times.

Below are some typical sequential logic properties, taking the SR-latch as an example.

Continuous value The output value of an SR-latch is continuously active or inactive.

Unidirectional The value of an SR-latch can only be changed by a set or reset input from the sending component.

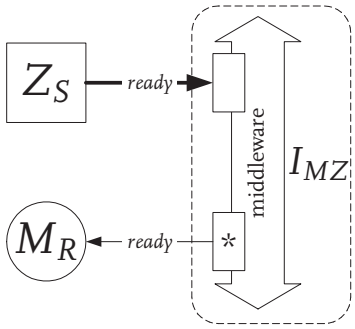
Set/reset A set or reset input results in an active or inactive latch output, respectively.

Although the SR-latch contains both discrete-event and continuous behavior, which could directly be modeled in hybrid χ, we restrict ourselves to the discrete-event version of χ, in which we abstract from the continuous behavior of the SR-latch. A discrete-event model of

the SR-latch is shown in Figure 5.4(a), in which the declaration and initialization of channels and variables are omitted for simplicity. The highest level of parallel composition (‖ on line 5) contains the processes of the sending component (lines 1–4) and the receiving component (lines 5–8) of the *ready_latch*, which indicates whether the sending component is ready for some next action. The sending process first sets the latch output to false (line 1) and later, when it is ready, to true (line 3). The receiving process waits until the other component is ready by using the latch value *ready* as a guard (line 5), which has to become true before the process continues. The discrete-event abstraction of the latch communication is modeled by adding another parallel process (line 6) to the model of the receiving component, i.e., on the second level of parallel composition (‖ on line 6). This additional process of the receiving component repetitively waits for new values of the *ready_latch* from the sending component. The variable *ready* is used to store the latest latch value and to share it with the other processes in the parallel composition of the receiving component. In this way, only the latest latch value is considered in the behavior of the receiving component.

```
1   ⟦   ( ready_latch ! false
2       ; ...
3       ; ready_latch ! true
4       )
5   ‖   ( ready → ...
6       ‖ ∗( ready_latch ? ready )
7       )
8   ⟧
```

(a) I_M



(b) I_{MZ}

Figure 5.4: I_M and I_{MZ} for SR-latch example

The properties given for the SR-latch are satisfied by the model since the *ready* variable always has a value (mimicking continuous behavior) and only the sending component can change the latch value by setting it to true (active) or by resetting it to false (inactive).

Figure 5.4(b) shows the implementation of I_{MZ} for the SR-latch example of Figure 5.4(a), with a realization of the sending component Z_S, a model of the receiving component M_R, and a topic *ready* for the SR-latch. Here, the publish-subscribe middleware is configured with different QoS properties than for the function call interaction type. For function calls, the queues in the connectors act as multi-message FIFO buffers from which the messages are removed after delivery to the receiving component. For the SR-latch, however, the queue in the connector for a receiving component should store and keep only the last value that is received from the sending component, similar to the behavior of the additional process on line 6 of Figure 5.4(a). This is achieved by configuring the publish-subscribe middleware with the QoS property 'keep one message as history', denoted with a * in Figure 5.4(b). This QoS property changes the behavior of the connector queue in such a way that only one message is stored in the queue, and that the message is not removed from the queue after delivery to the receiving component. Besides this difference in queue behavior for the

receiving component, the model connectors for the SR-latch interaction type are generated in the same way as for the function call interaction type. For a component realization that uses the SR-latch or another sequential logic interaction type (implemented in electronics), e.g., Z_S in Figure 5.4(b), the connector should adapt from software and electronic signals to publish-subscribe messages and vice versa. An example of such a realization connector is given in the industrial case study described in the next section.

In the described SR-latch example, only one value is stored (single-address memory). The infrastructure model I_M and its implementation I_{MZ} can easily be extended to represent multi-address memories as often used in memory mapped I/O and distributed I/O.

5.3 Practice: integration and testing of vacuum system model and real EUV source

The described approach on how to deal with infrastructure in the MBI&T method was applied to the EUV case study in order to answer QUESTION 1.3 on the practical applicability and profitability of model-based integration and system testing when only some component realizations are available. In Chapter 3, the focus of the EUV case study was on the functional behavior of the components and the system. In this chapter, however, we do not go into detail about the functional behavior; we now focus on the interaction behavior of the system and on the modeling, analysis, and implementation of the infrastructure.

Figure 5.5 recalls the components and interfaces involved in the EUV case study, abstracting from the physical parts and their physical interaction as shown in Figure 3.2. The vacuum system provides a function *goto_state* via which the control environment C_e of the system, e.g., a vacuum system operator or a higher level software controller, can instruct the vacuum system to go to either the vacuum or the vented state. To exchange information about their internal states, the vacuum system C_v and the source C_s are connected by an interface consisting of four SR-type latches.

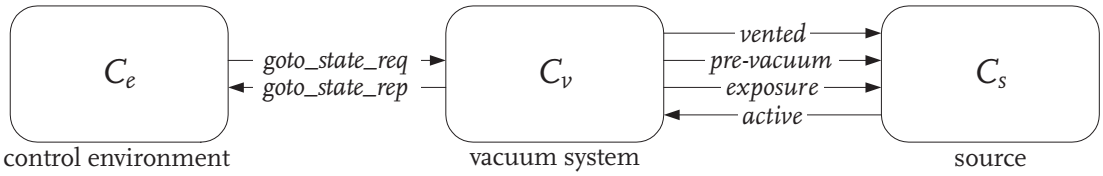


Figure 5.5: Components and interfaces involved in the EUV case study

The following subsections describe how the MBI&T activities of the procedure described in Section 2.3 were applied to the EUV case study. The focus is on the infrastructure, which is modeled as I_M (MBI&T activity 1b) to be included during system analysis (MBI&T activity 2), and implemented as I_{MZ} such that component models M_i can gradually be replaced by realizations Z_i (MBI&T activity 3b) for early system testing of the model-based integrated system (MBI&T activity 3c). Where necessary, we give a summary of the other MBI&T activities, which have been described in more detail in Chapter 3.

5.3.1 Modeling the components and their interaction in χ (activity 1)

In activity 1a of the MBI&T method, described in Chapter 3, the components shown in Figure 5.5 were modeled as the χ processes shown in Figure 5.6. The control environment was modeled as a single process M_e that can be configured to send requests and receive replies at certain points in time. The vacuum system M_v was modeled as a parallel composition of three processes, ($v1 \parallel v2 \parallel v3$), in which the core process $v1$ models the internal state behavior and processes $v2$ and $v3$ model the interaction with M_e and M_s, respectively. The source M_s was modeled as a parallel composition of four processes, ($s1 \parallel s2 \parallel s3 \parallel s4$), in which the core process $s1$ models the internal state behavior and processes $s2$, $s3$, and $s4$ model the latch interaction with M_v. The bold lines between processes of one component depict shared variables (two for the vacuum system and three for the source), which are used to exchange data between processes of one component.

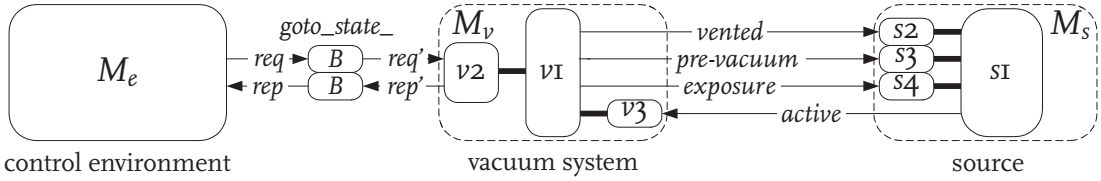


Figure 5.6: Process layout of the χ system model

In activity 1b, the interaction between the components was modeled using an infrastructure model I_M. The interaction between M_e and M_v is based on the non-blocking function call interaction type as explained in Subsection 5.2.1. The infrastructure model I_M for this interaction behavior is similar to Figure 5.3(a) with M_e as the client and process $v2$ of M_v as the server. Requests and replies of the *goto_state* function are modeled via the channels *goto_state_req* and *goto_state_rep*, respectively, using two buffer processes B to model the asynchronous communication. These buffer processes B were not discussed in Chapter 3, since at that point they were not important for the analysis of functional behavior.

The interaction between M_v and M_s is based on the SR-latch interaction type as explained in Subsection 5.2.2. The infrastructure model I_M for the SR-latches in the case study was already partly shown in Figure 3.6 in Chapter 3. In that figure, processes $s2$, $s3$, and $s4$ on lines 20, 21, and 22, respectively, were added to M_s to model the interaction for the 'vented', 'pre-vacuum', and 'exposure' latches from M_v to M_s. Process $v3$ is a similar process that was added to M_v to model the interaction for the 'active' latch from M_s to M_v.

5.3.2 Integrated system and infrastructure analysis (activity 2)

In Chapter 3, the functional behavior of the χ system model was analyzed by simulation of both nominal and exceptional scenarios and by UPPAAL model checking of formal properties derived from the system requirements R and the system design D. During this model-based system analysis, several design and integration problems were detected and fixed at an early stage of the development process. In this subsection, we only discuss the analysis of the infrastructural properties of the system model.

As described in Subsections 5.2.1 and 5.2.2, most of the function call and SR-latch properties are directly satisfied by the infrastructure model, independent of the system behavior. However, two function call properties depend on the system behavior and should be verified in the context of the system model. These properties, limited buffer size and consistency between the number of requests and replies, were verified in the case study in the following way. To detect buffer overflows, a boolean variable *overflow* was added to the model which becomes true whenever a buffer overflow is detected (line 3 of Figure 5.2). The limited buffer size property was verified by model checking the UPPAAL query *A*[ ] *not overflow*, which is only valid when *overflow* is false in all possible states of the model. To verify consistency, two counting variables, *nr_requests* and *nr_replies*, were added to the model and the UPPAAL query *A*[ ] $0 \leq$ *nr_requests - nr_replies* ≤ 1 was used for model checking. These two infrastructural properties of the function call interaction type were both satisfied for the system model developed in activity 1 of the case study.

5.3.3 Model-based integration and system testing (activities 3b and 3c)

In these activities of the case study, the source model M_s was replaced by its realization Z_s, i.e., the real EUV light source. This means that the interaction between the vacuum system model M_v and the source realization Z_s needed to be established. In the infrastructure realization I_Z, the latch communication between Z_v and Z_s is established via a multi-pin signal cable, of which four pins relate to the four latches. Since Z_s can only communicate with its environment via this multi-pin signal cable, the communication between M_v and Z_s should also be established via this multi-pin signal cable.

Integration of the control environment model M_e, the vacuum system model M_v and the source realization Z_s was achieved by using a model-based integration infrastructure I_{MZ} as shown in Figure 5.7(a), using a publish-subscribe middleware and appropriate component connectors. The interaction between M_e and M_v via the *goto_state* function was implemented in I_{MZ} by defining the topics *request* and *reply* and by configuring the published and subscribed topics in M_e and M_v accordingly, similar to Figure 5.3(b). For the interaction between M_v and Z_s, a topic was defined for each of the four latches and the connectors for the receiving components were configured with the QoS property 'keep one message as history', denoted with a * in Figure 5.7(a). In the figure, the arrows for the three SR-latches from M_v to Z_s ('vented', 'pre-vacuum', 'exposure') are combined and denoted by *3*latch*.

The middleware configuration described above and the model connectors for M_e and M_v were automatically generated from the χ system model using the χ toolset. The behavior of the message queues in the connectors was configured using the QoS properties corresponding to the interaction types, as described in the examples in Subsections 5.2.1 and 5.2.2.

The realization connectors, however, are case-specific and should be separately developed or, if available, selected from a library with generic and configurable connector templates. In the case study, the connector for Z_s should adapt the real latch communication via the multi-pin signal cable to the publish-subscribe communication used in the middleware and vice versa. To achieve this, a software/hardware adapter was used in the form of a remote I/O unit that allows different forms of analogous and digital input and output [Opto 22 2007; National Instruments 2007]. In the case study, we used a digital input module to receive

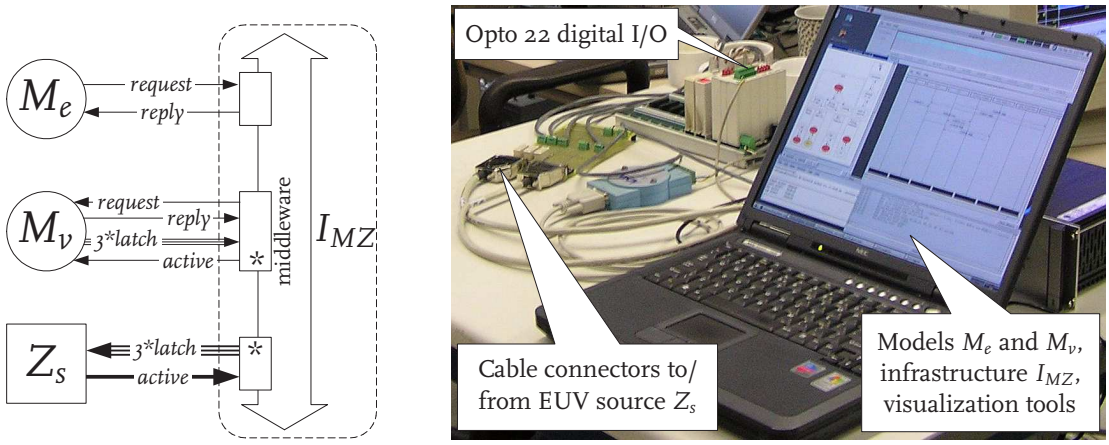


Figure 5.7: I_{MZ} for the case study

values of the 'active' latch and a digital output module to set values of the 'vented', 'pre-vacuum', and 'exposure' latches, see Figure 5.7(b) for a photo impression. Another aspect of I_{MZ} concerns its performance, i.e., the execution speed of the real-time χ simulator, the publish-subscribe middleware, and the connectors. In the case study, this execution speed was sufficient to properly handle the interaction between the vacuum system model and the source realization in time. In this interaction, the smallest amount of time between two consecutive events was in the order of multiple seconds. A more detailed investigation of the performance of I_{MZ} and its limits in applications with more time-critical interaction is left as future work.

Using the resulting model-based integration infrastructure I_{MZ}, we were able to test the model-based integrated system $\{M_e, M_v, Z_s\}_{I_{MZ}}$ significantly earlier (20 weeks before all realizations were available and integrated) and with less costs (no critical cleanroom time was needed as for real system testing). Similar to the simulation analysis in activity 2 of the case study, model M_e was configured with specific scenarios to test the model-based integrated system on different aspects, for both nominal and exceptional behavior. Creating non-nominal conditions to test exceptional behavior was easy in a model environment, whereas this may be quite difficult, time consuming, and risky when testing with realizations.

Besides showing the applicability of this MBI&T activity, the profitability also became clear because six integration problems were detected. The problems, which appeared to be caused by implementation errors in the source, could potentially lead to source damage (i.e., at least one full day of down time) and unnecessary waiting in the source (i.e., multiple hours of test time) during the real I&T phases in the ASML cleanroom. The models supported fast and easy diagnosis (i.e., determining the cause of an problem) and fixing of the detected implementation errors, as well as immediate retesting of the fixed system, reducing the diagnosis and fix time from hours to minutes. This means that the MBI&T activities probably saved several days of expensive cleanroom time during real integration and testing 20 weeks later (if the problems would remain undetected until that time). In the period af-

ter the model-based system tests, no additional problems related to the analyzed and tested aspects of the vacuum-source interaction were detected. This means that, because of the MBI&T method, the system quality was improved at an early stage, preventing late and thus expensive fixing during real system integration and testing.

The total amount of time used for testing, diagnosis, fixing, and retesting of the model-based integrated system was significantly lower than the estimated amount of time that would be required to perform the same tests on the real system: one half of a day against four days. This time reduction, which also indicates a test phase improvement with an increased risk reduction rate as proposed in Section 1.4, is caused by several factors. First, experience in real system testing shows that setting up the system for testing may consume a considerable amount of time. In the case study, for example, a certain test may require that the initial vacuum system state is vented while the end state of the previous test was vacuum. This also holds for the re-execution of tests that change the system state, e.g., a test that starts in the vented state and ends in the vacuum state. In model-based system testing, less test setup time is required because setting up a model to another initial state usually boils down to changing some variables, e.g., changing the initial value of the vacuum system state variable. Second, testing with realizations may also suffer from time lost on solving minor system problems that are unimportant for the tests. In the case study, for example, the real vacuum system contains many potential problem sources (e.g., a malfunctioning sensor or valve) that could result in a system that is unable to initialize, thus prohibiting test execution. Model-based system testing does not suffer from this issue, since the models only contain the behavior that is important for the tests and abstract from the minor problems that potentially prohibit test execution. Third, the use of models for testing reduces the time spent on diagnosis of problems when compared to real system testing. On the one hand, the number of sources that could potentially cause a problem is reduced since the models only contain the behavior that is important for the tests, i.e., abstracting from all other components and aspects which form potential problem sources in real system testing. On the other hand, the complete insight in and control over the models makes the distinction between the potential problem sources more clear. Note that diagnosis in the case study, although supported and improved by the models, was still performed manually. As investigated in another part of the TANGRAM project [Pietersma and van Gemund 2007; Tretmans 2007], models can also be used for automated diagnosis.

5.4 Conclusions

This chapter described an approach to model, analyze, and implement component interaction in the MBI&T method, using a modeling language with synchronous communication, such that the infrastructural properties analyzed using the system model remain valid during the integration of models and realizations. In the presented approach, the behavior of the (asynchronous) infrastructure realization I_Z, based on the 'as is' system design D, is modeled as infrastructure model I_M using synchronous communication, e.g., in the form of message buffers between components. This infrastructure model I_M is included in the system model in order to analyze properties related to infrastructural and system behavior,

e.g., by simulation and model checking. Subsequently, the infrastructure as designed in D and modeled in I_M is implemented in a model-based integration infrastructure I_{MZ}, using a publish-subscribe middleware and appropriate connectors, allowing easy integration and testing of combined models and realizations. Using this approach in the MBI&T method enables early integration and system testing when only some component realizations are available, which satisfies the requirements described in Subsection 2.3.3 and answers QUESTION 1.2.

The described industrial applications, which concerned examples of two common interaction types and the EUV case study, provide a positive answer to QUESTION 1.3: it is feasible and profitable to apply the MBI&T method in current industrial practice when only some component realizations are available. In these industrial applications, the transition from synchronous process algebraic models to distributed asynchronous realizations proved to be straightforward. The considered asynchronous interaction types (function calls and SR-latches) can easily be modeled in a synchronous modeling language such as χ, using additional processes in the system model. These synchronous system models provide a good understanding of system behavior and enable verification of properties related to both infrastructural and system behavior. The publish-subscribe middleware and the connectors provide a simple means to implement the modeled interaction behavior, allowing easy integration of models and realizations. Here, the interaction between models and realizations and the real-time behavior are no longer handled according to the formal model semantics. The handling of component interaction and real-time behavior should be implemented such that the overall system behavior is equivalent to the modeled and analyzed system behavior, possibly with some acceptable restrictions, e.g., regarding the execution speed. For example, different QoS properties of the middleware can be used to influence the behavior of the connector queues in accordance with the modeled behavior, e.g., the 'keep one message as history' QoS property for the SR-latch interaction type. For the industrial examples in this chapter, we gave an informal and intuitive indication that the correctness of the infrastructural properties is preserved when replacing I_M with I_{MZ}. The described approach can be applied to other interaction types in a similar way.

The EUV case study showed that the χ toolset, including the real-time χ simulator and the publish-subscribe middleware based on JABBER, is suitable for the integration and testing of combined models and realizations, which means that, for this MBI&T activity, the hypothesis on the applicability of the χ toolset is accepted. Furthermore, the modeling and analysis activities in the case study, as described in Chapter 3 and this chapter, showed relevant advantages for the system development process. In terms of the T-Q-C business drivers, the modeling activities helped to clarify, correct, and complete the design documentation, improving the product quality Q. By simulation and verification, several design and integration problems were detected and fixed earlier (i.e., reduced time-to-market T) and with less costs C when compared to current system development. Finally, the integration of models and realizations using the model-based integration infrastructure enabled us to perform system tests earlier (several months before the real I&T phases), faster, and with lower costs. During the system tests, multiple implementation errors in the realization were detected, immediately fixed, and retested, saving significant amounts of time and rework during the real I&T phases and thus reducing time-to-market T. Here, the complete insight in and control over

the models and the (non-nominal) test conditions improved the test execution process, i.e., also the risk reduction rate increased as proposed by the second solution in Section 1.4.

CHAPTER 6

Integration and testing process

Where and when to apply models?

Chapters 3, 4, and 5 described the MBI&T activities as introduced in Chapter 2, and showed the applicability and profitability of each activity in practice. For the I&T process, which is the focus of this chapter, these MBI&T activities form new possibilities to reduce time-to-market T or to increase the product quality Q. In contrast to these benefits for the I&T process, the MBI&T method also introduces additional costs C, e.g., time is needed to model the components. These investments in modeling need to be made before the actual benefits become clear, often without knowing if and to which extent the benefits outweigh the costs. In some cases, the investments in modeling are profitable, e.g., when the realization of a component is available only late in the development process or when testing with realizations is expensive. In other cases, it is wise not to invest in models but to perform the tests with realizations only, e.g., for mature or low risk components.

Making decisions on whether or not to use models for integration and testing can be supported by estimations on the involved risk in the system, on the development or delivery times and the availability of realizations, and on the costs of testing with realizations. In current industrial practice, the decision making process is usually based on personal intuition and experience. In this chapter, we introduce a quantitative decision making process that takes the costs C into account to determine where (QUESTION 2.1) and when (QUESTION 2.2) the I&T process can profit from models.

This chapter is mainly based on [Braspenning et al. 2007d,a], in which co-author Boumen provided the I&T sequencing techniques, and is organized as follows. Section 6.1 first describes the current I&T process and distinguishes nine categories of I&T activities. Subsequently, this section shows how the MBI&T method and techniques are applied to the current I&T process and how each of the nine categories can be supported by models, answering QUESTION 2.1. Section 6.2 shows how the integration and test sequencing method from [Boumen 2007] can be used to quantify the costs of various I&T processes in order to decide when it is profitable to apply models in the I&T process, answering QUESTION 2.2. An industrial application of this quantitative decision making process is shown in Section 6.3, which answers QUESTION 2.3 on the practical applicability. Finally, the conclusions are given in Section 6.4.

6.1 Current and model-based I&T process

This section first describes the current I&T process such as used at ASML in more detail. Subsequently, QUESTION 2.1, repeated below, is answered by showing which activities in the current I&T process can be supported by models.

> **QUESTION 2.1** Which activities in the current I&T process can be supported by the early I&T method?

As an example, we use an I&T process that is common for many high-tech multi-disciplinary systems: upgrading a system with new hardware and software to improve the system performance, e.g., adding a new sensor with accompanying control software to improve the measurement accuracy. In this example, the goals of integration and testing are to show the functionality and performance of the system upgrade as soon as possible, and to show that the system upgrade does not negatively affect the functionality and performance of the original system. We show how such a system upgrade is dealt with in both the current I&T process and the model-based I&T process.

6.1.1 Current I&T process

Let us consider an existing system that consists of several hardware and software components. The system is upgraded by implementing some new or improved functionality, which is denoted by a delta sign (Δ). To implement this Δ-functionality, certain components of the original system need to be upgraded, or new components need to be developed and added to the system. Similar to Figure 2.2 in Chapter 2, our view on the development process of this Δ-functionality starts with the global requirements R_Δ and design D_Δ, as shown on the left-hand side of Figure 6.1. After that, the software and hardware components for the Δ-functionality, denoted by ΔSW and ΔHW, respectively, are separately developed, starting with the requirements $R_{\Delta SW}$ and $R_{\Delta HW}$, followed by the designs $D_{\Delta SW}$ and $D_{\Delta HW}$ and, finally, the realizations $Z_{\Delta SW}$ and $Z_{\Delta HW}$.

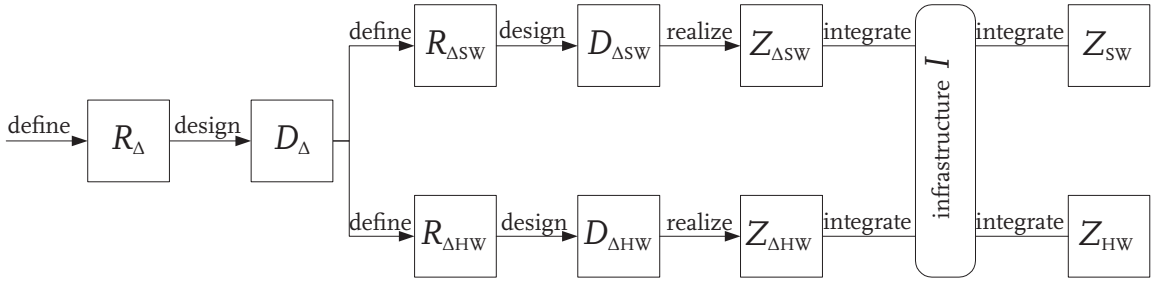


Figure 6.1: Current development and integration of a Δ-functionality

The right-hand side of Figure 6.1 shows the integration of the Δ components $Z_{\Delta SW}$ and $Z_{\Delta HW}$ with the software and hardware components of the original system, considered

here as one software component Z_{SW} and one hardware component Z_{HW}, respectively. The four components are integrated by means of some infrastructure I. In this chapter, we abstract from the different forms of infrastructure as explained in Chapter 5 and only consider the 'generalized' infrastructure I.

The four component realizations $Z_{\Delta SW}$, $Z_{\Delta HW}$, Z_{SW} and Z_{HW} can be integrated and tested in many ways during the I&T process. Within the current I&T process at ASML, nine different categories of I&T activities can be distinguished, which focus on different aspects of the components or the system and require different combinations of realized and integrated components. These nine categories are listed in Table 6.1, where the integration of component realizations Z_i and Z_j by means of infrastructure I is denoted as $\{Z_i, Z_j\}_I$.

Table 6.1: Categories of I&T activities in current I&T process

Category	Required components	Explanation
1. SW qualification testing	$\{Z_{SW}, Z_{HW}\}_I$ and later $\{Z_{\Delta SW}, Z_{SW}, Z_{HW}\}_I$	Periodic qualification of the so-called 'qualified baseline' (QBL) [Horch 2003], a common repository for all new software developments that supports all machine types, by testing it on a set of representative hardware systems Z_{HW}
2. SW component testing	$Z_{\Delta SW}$	Testing the new software component in isolation
3. SW integration testing	$\{Z_{\Delta SW}, Z_{SW}\}_I$	Testing the new software component in combination with the original software system Z_{SW}
4. SW regression testing	$\{Z_{\Delta SW}, Z_{SW}, Z_{HW}\}_I$	Testing whether any of the original system functions are negatively affected by the new software component, performed on the original hardware system Z_{HW}
5. HW component testing	$Z_{\Delta HW}$	Testing the new hardware component in isolation
6. HW integration testing	$\{Z_{\Delta HW}, Z_{HW}\}_I$	Testing the new hardware component in combination with the original hardware system Z_{HW}
7. Δ-functionality test bench testing	$\{Z_{\Delta SW}, Z_{SW}, Z_{\Delta HW}\}_I$	Testing the new Δ-functionality (also called progression testing) on a 'test bench', i.e., a partial hardware system including the new hardware component $Z_{\Delta HW}$, used for development tests
8. Δ-functionality system testing	$\{Z_{\Delta SW}, Z_{SW}, Z_{\Delta HW}, Z_{HW}\}_I$	Testing the new Δ-functionality on a complete system, i.e., Z_{HW} upgraded with $Z_{\Delta HW}$
9. System testing	$\{Z_{\Delta SW}, Z_{SW}, Z_{\Delta HW}, Z_{HW}\}_I$	Testing the functionality and performance of the complete system after all Δ-functionalities are integrated and tested, before system shipment

Figure 6.2 shows a typical I&T process for a system upgrade. From left to right, the sequence of I&T activities, denoted by vertical lines, is shown. The numbers correspond to the nine categories of I&T activities in Table 6.1, and the dots indicate which components are integrated and tested. The horizontal lines depict the lifetime of each component: a

dashed line means that the component is being developed; a flag symbol followed by a solid line means that the component realization is available. The flag symbols and the letters indicate the following milestones: (a) QBL Z_{SW} passes qualification tests; (b) development of $Z_{\Delta SW}$ and $Z_{\Delta HW}$ is started, possibly based on the original system (denoted by dashed upward arrows); (c) $Z_{\Delta SW}$ is available; (d) $Z_{\Delta SW}$ passes software tests and is integrated in the QBL Z_{SW} (denoted by downward arrow); (e) upgraded QBL $\{Z_{\Delta SW}, Z_{SW}\}_I$ passes qualification tests; (f) $Z_{\Delta HW}$ is available; (g) $Z_{\Delta SW}$ and $Z_{\Delta HW}$ pass test bench tests; (h) $Z_{\Delta HW}$ passes hardware integration tests and the hardware system Z_{HW} is upgraded to $\{Z_{\Delta HW}, Z_{HW}\}_I$ (denoted by downward arrow); (i) similar to the depicted Δ-functionality, the other Δ-functionalities are integrated and tested; (j) complete system with all Δ-functionalities passes tests and is shipped to customer. Note that the figure only shows a sequence and does not contain information on the possible start times and durations of the activities. For example, in the case that the hardware upgrade is available earlier than the software upgrade (i.e., milestone f before b), I&T activities 5 and 6 could be performed before I&T activities 2, 3, and 4.

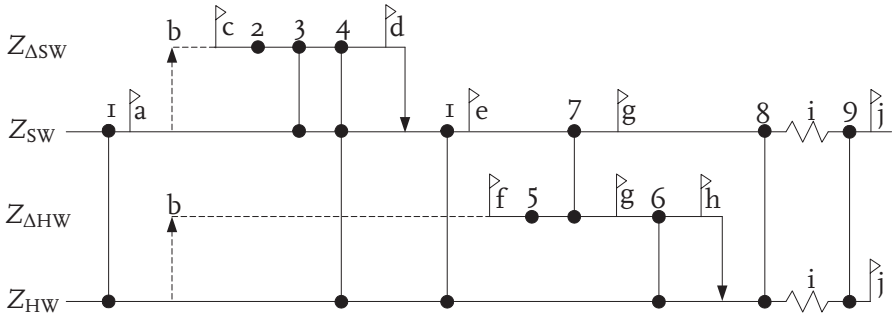


Figure 6.2: Typical I&T process for the system upgrade example

Note that by removing particular components and related I&T activities from the system upgrade example, other I&T processes can also be represented. For example, a software only I&T process is obtained by removing the hardware components and all I&T activities that involve hardware. The I&T process of a completely new system is obtained by removing the original system software and hardware, implying that there is no initial software QBL, and that the first software component that is realized becomes the software QBL.

The main disadvantage of the current I&T process is that the I&T activities can only be performed when the realizations are available. Especially for testing on the system level (categories 7, 8, and 9) this is problematic, because it means that feedback on the system behavior and performance is obtained late in the process, where fixing the problems is expensive. The previous chapters of this thesis showed that early integration and testing is possible by using models in the I&T process. The remainder of this section shows which activities of the I&T process can be supported by models, using the same system upgrade example.

6.1.2 Model-based I&T process

Figure 6.3 shows the development and integration of a Δ-functionality using the MBI&T method, with models $M_{\Delta SW}$, $M_{\Delta HW}$, and M_{HW} of the Δ software component, the Δ hardware

component, and the original complete hardware system, respectively. The reason for having M_{HW} but not M_{SW} is explained later. The choice of integrating either the model or the realization of a component, or none of them, is depicted by the integration 'switches'. In contrast to the current I&T activities, in which only realizations are used, the MBI&T activities from Chapters 3 through 5 can be performed with models instead of realizations, which has several advantages as shown in these chapters. Regarding the costs C of testing, many of the MBI&T activities can be performed on a common computer system using modeling and analysis software tools. The test costs in such a desktop environment are generally lower than the costs of realization tests, which, in the case of ASML, may require expensive machine time and cleanroom facilities.

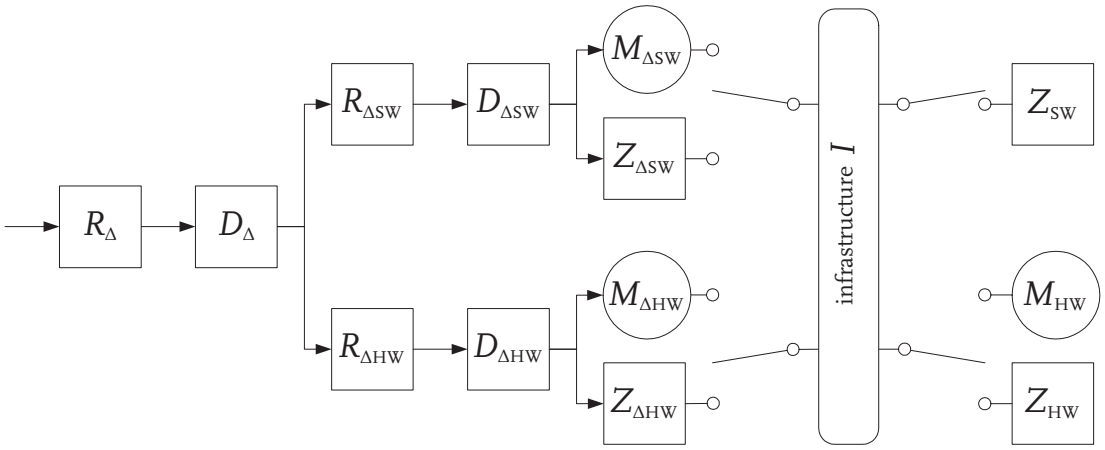


Figure 6.3: Development and integration of a Δ-functionality in the MBI&T method

Although models can support all nine categories of I&T activities as shown in Table 6.1, they cannot fully replace testing with realizations, since models are always abstractions of reality and usually do not cover all aspects of a component. For example, the models in the EUV case study used throughout this thesis focus on component interaction and time behavior, which are suitable to test and detect problems in these aspects but unsuitable to test other aspects such as the quality of the EUV light. Sooner or later, when they become available, the realizations of the components and the system will be used to test at least the remaining aspects. However, these realization tests can probably be performed faster and with less costs, since the MBI&T activities already reduced some risk by detecting and preventing several problems. This was also experienced in the EUV case study as described in Chapter 5: in the period after the successful MBI&T activities on the source realization, no further problems were detected in the component interaction and time behavior, i.e., all test time could be spent on other aspects such as the quality of the EUV light.

In the current I&T process of ASML, the software qualification tests (category 1) consume quite some machine time, approximately one full day of testing each week. Besides that machine time is limited and expensive, experience shows that also setting up the system for testing may consume a considerable amount of time. Moreover, much time may be lost on solving minor machine problems that are unimportant for the tests, e.g., a malfunctioning

sensor that is not involved in the test but prevents the system to initialize. Test time and costs may be reduced by using hardware models instead of hardware realizations for certain parts of the qualification tests. For example, the qualification of the system throughput in principle depends only on the sequence and durations of all hardware actions. When the durations of these hardware actions are modeled as time delays in a model M_{HW} of the hardware system Z_{HW}, and when the software Z_{SW} executes the sequence of actions on the model M_{HW}, the system throughput can be qualified without a hardware realization Z_{HW}. In this way, software qualification tests can be performed in a low cost desktop environment with a hardware model M_{HW}, using $\{Z_{SW}, M_{HW}\}_I$ instead of $\{Z_{SW}, Z_{HW}\}_I$. Furthermore, models require less test setup time, and they do not suffer from the minor problems that may occur in other components not involved in the tests, since the hardware model only contains the behavior important for the tests and abstracts from these problems. When hardware models M_{HW} are used for periodic software qualification testing, it is important that these models are continuously maintained and qualified as well, which can be achieved by establishing a QBL for the hardware models M_{HW}, similar to the software QBL Z_{SW}. As long as the hardware models are correct representations of the hardware realizations for the aspects covered by the software qualification tests, they can safely replace the realizations for these tests.

For software component and integration testing (categories 2 and 3), a model $M_{\Delta SW}$ of the Δ software component can be used as replacement of $Z_{\Delta SW}$. As shown in Chapters 3 and 5, (real-time) simulation techniques can be used to analyze the behavior of the model in isolation or in combination with the other software Z_{SW}. When the realization $Z_{\Delta SW}$ is available, automatic model-based testing can be used to determine its conformance to the model $M_{\Delta SW}$, as shown in Chapter 4. The model $M_{\Delta SW}$ can also replace $Z_{\Delta SW}$ for software regression testing (category 4), which can be performed either on the real hardware system Z_{HW} or on the hardware model M_{HW}, in a way similar to model-based software qualification testing (category 1). To enable these MBI&T activities, the software components directly related to the tested component may also be modeled as its environment. In principle, such an environment model is a partial model of the original software Z_{SW}, which could be denoted by M_{SW}. However, we expect that continuous development, maintenance, and qualification of a complete model M_{SW} (as is done for M_{HW} in category 1) is not necessary. Since testing with Z_{SW} is also performed in a desktop environment, the cost advantage of using a model M_{SW} instead of Z_{SW} will probably be lower than for M_{HW}. This means that the effort invested in maintaining and qualifying M_{SW} will be less profitable, and that it is better to directly use Z_{SW} when the complete software is required for testing. Nevertheless, parts of M_{SW} can and will still be used as environment model for other MBI&T activities.

Similar to its software counterpart, a model $M_{\Delta HW}$ can support component and integration testing of the Δ hardware component (categories 5 and 6) by means of simulation, automatic model-based testing, and model-based analysis of the upgraded hardware model $\{M_{HW}, M_{\Delta HW}\}_I$.

Testing the complete Δ-functionality using a test bench (category 7) can be supported by four MBI&T activities. First, the Δ-functionality can be tested by using the integrated models of the Δ components, i.e., $\{M_{\Delta SW}, M_{\Delta HW}\}_I$, in which $M_{\Delta HW}$ is a model of test bench $Z_{\Delta HW}$. Since only models are used in this test, model-based analysis techniques such as model

checking can be used for exhaustive analysis of all possible behaviors of the system model, as shown in Chapter 3. Second, the model $M_{\Delta SW}$ can be integrated with the other software Z_{SW}, and tested on the test bench model $M_{\Delta HW}$. Third, the realization of the upgraded software system, i.e., $\{Z_{\Delta SW}, Z_{SW}\}_I$, can be tested on the model of the test bench $M_{\Delta HW}$. Finally, in the case that $Z_{\Delta HW}$ is available before the software realization $Z_{\Delta SW}$, the model $M_{\Delta SW}$ can be tested with Z_{SW} on the test bench realization $Z_{\Delta HW}$. The four MBI&T activities that support the Δ-functionality system testing (category 8) are similar to those of category 7, but now they involve the complete hardware system instead of the test bench only, i.e., including M_{HW} or Z_{HW}, depending on whether $M_{\Delta HW}$ or $Z_{\Delta HW}$ is used, respectively. Similar to software qualification testing (category 1), system testing (category 9) can be supported by a model of the (upgraded) hardware, replacing the hardware realization to partially test the functionality and performance of the system.

The following list summarizes and identifies all possible I&T activities for each category, including both the current I&T activities from Table 6.1 and the new MBI&T activities as described above, thus answering QUESTION 2.1. For each of the nine categories, the I&T activities with realizations only are marked with a 'Z', and the MBI&T activities are marked with an 'M', followed by a letter in the case of multiple MBI&T activities.

1. Software qualification testing:

1Za: $\{Z_{SW}, Z_{HW}\}_I$
1Ma: $\{Z_{SW}, M_{HW}\}_I$
1Zb: $\{Z_{\Delta SW}, Z_{SW}, Z_{HW}\}_I$
1Mb: $\{Z_{\Delta SW}, Z_{SW}, M_{HW}\}_I$

2. Software component testing:

2Z: $Z_{\Delta SW}$
2Ma: $M_{\Delta SW}$
2Mb: $Z_{\Delta SW}$ vs. $M_{\Delta SW}$

3. Software integration testing:

3Z: $\{Z_{\Delta SW}, Z_{SW}\}_I$
3M: $\{M_{\Delta SW}, Z_{SW}\}_I$

4. Software regression testing:

4Z: $\{Z_{\Delta SW}, Z_{SW}, Z_{HW}\}_I$
4Ma: $\{M_{\Delta SW}, Z_{SW}, M_{HW}\}_I$
4Mb: $\{M_{\Delta SW}, Z_{SW}, Z_{HW}\}_I$

5. Hardware component testing:

5Z: $Z_{\Delta HW}$
5Ma: $M_{\Delta HW}$
5Mb: $Z_{\Delta HW}$ vs. $M_{\Delta HW}$

6. Hardware integration testing:

6Z: $\{Z_{\Delta HW}, Z_{HW}\}_I$
6M: $\{M_{\Delta HW}, M_{HW}\}_I$

7. Δ-functionality test bench testing:

7Z: $\{Z_{\Delta SW}, Z_{SW}, Z_{\Delta HW}\}_I$
7Ma: $\{M_{\Delta SW}, M_{\Delta HW}\}_I$
7Mb: $\{M_{\Delta SW}, Z_{SW}, M_{\Delta HW}\}_I$
7Mc: $\{Z_{\Delta SW}, Z_{SW}, M_{\Delta HW}\}_I$
7Md: $\{M_{\Delta SW}, Z_{SW}, Z_{\Delta HW}\}_I$

8. Δ-functionality system testing:

8Z: $\{Z_{\Delta SW}, Z_{SW}, Z_{\Delta HW}, Z_{HW}\}_I$
8Ma: $\{M_{\Delta SW}, M_{\Delta HW}, M_{HW}\}_I$
8Mb: $\{M_{\Delta SW}, Z_{SW}, M_{\Delta HW}, M_{HW}\}_I$
8Mc: $\{Z_{\Delta SW}, Z_{SW}, M_{\Delta HW}, M_{HW}\}_I$
8Md: $\{M_{\Delta SW}, Z_{SW}, Z_{\Delta HW}, Z_{HW}\}_I$

9. System testing:

9Z: $\{Z_{\Delta SW}, Z_{SW}, Z_{\Delta HW}, Z_{HW}\}_I$
9M: $\{Z_{\Delta SW}, Z_{SW}, M_{\Delta HW}, M_{HW}\}_I$

Figure 6.4 shows all I&T activities of the MBI&T process, in a way similar to Figure 6.2. The flag symbols and the letters indicate the following milestones: (a) QBL Z_{SW} passes qualification tests; (b) modeling of $M_{\Delta SW}$ and $M_{\Delta HW}$ is started, possibly based on the original system (denoted by dashed upward arrows); (c) $M_{\Delta SW}$ and $M_{\Delta HW}$ are available; (d) $M_{\Delta SW}$ and $M_{\Delta HW}$ pass model tests; (e) development of $Z_{\Delta SW}$ and $Z_{\Delta HW}$ is started, possibly based on the original system and the models (denoted by dashed upward arrows); (f) $Z_{\Delta SW}$ is available; (g) $Z_{\Delta SW}$ passes software tests (model-based component testing, i.e., testing realization against model, is denoted by a double headed arrow) and is integrated in the QBL Z_{SW} (denoted by downward arrow); (h) upgraded QBL $\{Z_{\Delta SW}, Z_{SW}\}_I$ passes qualification tests; (i) $Z_{\Delta HW}$ is available; (j) $Z_{\Delta SW}$ and $Z_{\Delta HW}$ pass test bench tests; (k) $Z_{\Delta HW}$ passes hardware integration tests and both Z_{HW} and M_{HW} are upgraded (denoted by downward arrows); (l) similar to the depicted Δ-functionality, other Δ-functionalities are integrated and tested; (m) complete system with all Δ-functionalities passes system tests and is shipped to the customer.

As an example, the circles indicate the I&T activities of category 7, Δ-functionality test bench testing. Their positions in the development process clearly illustrate how models enable earlier testing on the system level when compared to the current I&T process, in which only the realization test 7Z can be performed late in the process.

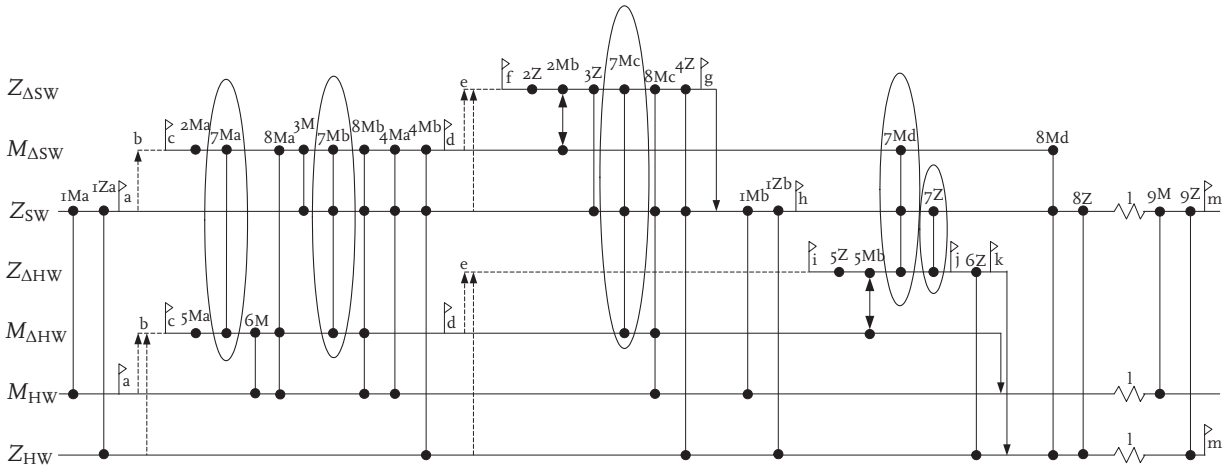


Figure 6.4: Model-based integration and testing process

Although Figure 6.4 shows more I&T activities than Figure 6.2, the number of activities does not relate to the total duration of the I&T process, because the possible start times and the durations of the I&T activities are not included. For example, in the case that the hardware upgrade is realized earlier than the software upgrade (i.e., milestone i before f), the MBI&T activities 7Md and 8Mc could be performed before 2Z.

6.2 Theory: integration and test sequencing

Although all possible (model-based) I&T activities have been defined in the previous section, there is still a problem that needs to be addressed before the MBI&T process can be applied

to a real I&T problem. This problem, which also exists in the current I&T process, is called integration and test sequencing, which was investigated in another part of the TANGRAM project [Boumen 2007; Tretmans 2007]. Integration and test sequencing involves making decisions on which components should be integrated when, and which tests should be performed in which order on which components. These decisions result in a sequence of I&T activities that can be optimized towards criteria such as lead time, total test time, test costs, and remaining risk (i.e., quality) in the system, depending on the importance of the T-Q-C business drivers as explained in Chapter 1.

For the MBI&T process, there is not only the problem of deciding on the sequence of the I&T activities, but there is also a choice whether or not to use models for certain I&T activities, as expressed in QUESTION 2.2, which is repeated below.

QUESTION 2.2 When is it profitable to apply the early I&T method?

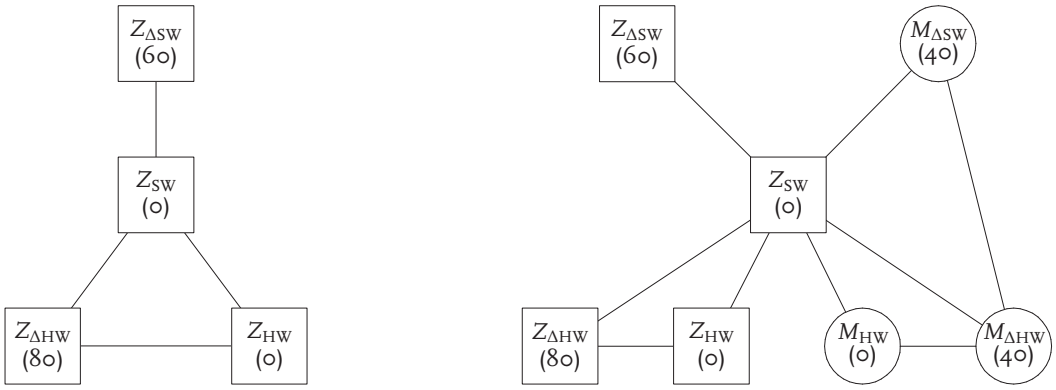
Answering QUESTION 2.2 involves a trade-off between the potential benefits of applying the MBI&T method (e.g., shorter time-to-market T and improved product quality Q) and the additional costs C needed to enable the MBI&T activities (e.g., time needed to model and integrate the components). We use the I&T sequencing method from [Boumen 2007] to determine the (nearly) optimal I&T sequences and to quantify the related costs for various I&T processes. By comparing the costs of I&T processes with and without models, this provides a quantitative decision making process to decide when it is profitable to use models for integration and testing.

In this section, we introduce this quantitative decision making process by giving an illustrative example that is based on the system upgrade I&T process from the previous section. The system upgrade example is instantiated once with realizations only (as in the current I&T process), and once with the possibility to use models as well (as in the MBI&T process), such that the resulting I&T sequences and related costs can be compared.

The input for the I&T sequencing method is an I&T process model that contains an abstract representation of an I&T problem. An I&T process model defines properties of and relations between the components and interfaces of a system, the potential faults related to the components and interfaces, and the tests that can be performed on certain combinations of integrated components to detect certain faults.

Figure 6.5 and Table 6.2 show the information used for the I&T process models of the system upgrade example. Figures 6.5(a) and 6.5(b) depict the components (boxes for realizations and circles for models) and interfaces (lines) for the current and for the model-based I&T process, respectively. The numbers between the brackets denote the development or delivery times of each component. This example uses fictitious but representative development times for the system upgrade I&T process, in which the original hardware and software are available from the start (development time is zero) and the Δ software component is available after 60 time units, which is 20 time units before the Δ hardware component. In the MBI&T process, the models of the Δ components are available after 40 time units. Note that an interface can also indicate a possible integration of components that is only needed for a certain I&T activity. For example, the interface between Z_{SW} and $Z_{\Delta HW}$ is only needed for test bench testing and does not necessarily exist in the actual system. Also note that the model-based

interface layout is symmetrical except for the additional interface between $M_{\Delta SW}$ and $M_{\Delta HW}$. This interface is used for model-based analysis of the Δ-functionality (I&T activity 7Ma), which can be performed early in the I&T process without realizations (see Figure 6.4).



(a) Current I&T process (b) MBI&T process

Figure 6.5: Components and interfaces for the system upgrade example

Table 6.2 shows the available tests in the I&T process model of the system upgrade example, including the components that need to be available and integrated for each test, and the test durations. The tests in the table relate to all 29 I&T activities of the nine categories listed in the previous section, i.e., including both the current and the model-based I&T activities. In the I&T process model, the choice of using models or realizations for a certain test can be expressed in several ways. For the system upgrade example, we express it in a simple way which is sufficient to explain the quantitative decision making process in this section. In the industrial application described in the next section, we use a more detailed and more realistic way of expressing this choice.

For the system upgrade example, we subdivide all tests into tests that can be performed with realizations only (denoted with a 'Z' in Table 6.2) and into tests that can possibly be performed with models (denoted with an 'M' in Table 6.2). This subdivision is done for both the current and model-based I&T process, under the assumption that the aspects covered by an 'M' test can also be tested using realizations, i.e., also in the current, non model-based, I&T process. This assumption is valid in most cases since the models are abstract representations of the realizations and their behavior, implying that the tested behavior of a model will also occur in the corresponding realization. As a result, the 'M' tests contain a choice of using realizations as an alternative to the models, which is denoted by the round brackets in the second column in Table 6.2. For the current I&T process, only the realization alternatives can be chosen, while for the MBI&T process, both alternatives can be chosen. In this way, we can use a single set of tests to compare the current and model-based I&T process. Taking category 1 as an example, we see that I&T activity 1Za requires Z_{SW} and Z_{HW} to be available and integrated. I&T activity 1Ma always requires Z_{SW}, but gives a choice to use either the hardware system model M_{HW} or the realization Z_{HW}. For the current I&T process, only the equivalent realization test with Z_{HW} can be used for 1Ma, while in the MBI&T process, the model M_{HW} can also be used for 1Ma if this is profitable.

Table 6.2: Available tests for the system upgrade example

Test	Required components	Time
1Za	$\{Z_{SW}, Z_{HW}\}_I$	6
1Ma	$\{Z_{SW}, (M_{HW}/Z_{HW})\}_I$	2
1Zb	$\{Z_{\Delta SW}, Z_{SW}, Z_{HW}\}_I$	6
1Mb	$\{Z_{\Delta SW}, Z_{SW}, (M_{HW}/Z_{HW})\}_I$	2
2Z	$Z_{\Delta SW}$	1
2Ma	$(M_{\Delta SW}/Z_{\Delta SW})$	1
2Mb	$Z_{\Delta SW}$ vs. $M_{\Delta SW}$	1
3Z	$\{Z_{\Delta SW}, Z_{SW}\}_I$	2
3M	$\{(M_{\Delta SW}/Z_{\Delta SW}), Z_{SW}\}_I$	1
4Z	$\{Z_{\Delta SW}, Z_{SW}, Z_{HW}\}_I$	3
4Ma	$\{(M_{\Delta SW}/Z_{\Delta SW}), Z_{SW}, (M_{HW}/Z_{HW})\}_I$	1
4Mb	$\{(M_{\Delta SW}/Z_{\Delta SW}), Z_{SW}, Z_{HW}\}_I$	2
5Z	$Z_{\Delta HW}$	2
5Ma	$(M_{\Delta HW}/Z_{\Delta HW})$	1
5Mb	$Z_{\Delta HW}$ vs. $M_{\Delta HW}$	1
6Z	$\{Z_{\Delta HW}, Z_{HW}\}_I$	3
6M	$\{(M_{\Delta HW}/Z_{\Delta HW}), (M_{HW}/Z_{HW})\}_I$	1
7Z	$\{Z_{\Delta SW}, Z_{SW}, Z_{\Delta HW}\}_I$	4
7Ma	$\{(M_{\Delta SW}/Z_{\Delta SW}), (M_{\Delta HW}/Z_{\Delta HW})\}_I$	2
7Mb	$\{(M_{\Delta SW}/Z_{\Delta SW}), Z_{SW}, (M_{\Delta HW}/Z_{\Delta HW})\}_I$	1
7Mc	$\{Z_{\Delta SW}, Z_{SW}, (M_{\Delta HW}/Z_{\Delta HW})\}_I$	2
7Md	$\{(M_{\Delta SW}/Z_{\Delta SW}), Z_{SW}, Z_{\Delta HW}\}_I$	1
8Z	$\{Z_{\Delta SW}, Z_{SW}, Z_{\Delta HW}, Z_{HW}\}_I$	4
8Ma	$\{(M_{\Delta SW}/Z_{\Delta SW}), (M_{\Delta HW}/Z_{\Delta HW}), (M_{HW}/Z_{HW})\}_I$	2
8Mb	$\{(M_{\Delta SW}/Z_{\Delta SW}), Z_{SW}, (M_{\Delta HW}/Z_{\Delta HW}), (M_{HW}/Z_{HW})\}_I$	2
8Mc	$\{Z_{\Delta SW}, Z_{SW}, (M_{\Delta HW}/Z_{\Delta HW}), (M_{HW}/Z_{HW})\}_I$	2
8Md	$\{(M_{\Delta SW}/Z_{\Delta SW}), Z_{SW}, Z_{\Delta HW}, Z_{HW}\}_I$	2
9Z	$\{Z_{\Delta SW}, Z_{SW}, Z_{\Delta HW}, Z_{HW}\}_I$	60
9M	$\{Z_{\Delta SW}, Z_{SW}, (M_{\Delta HW}/Z_{\Delta HW}), (M_{HW}/Z_{HW})\}_I$	20

The test durations in the third column of Table 6.2 are fictitious but give a representative distribution of the test time over all I&T activities of the system upgrade I&T process. For simplicity, this example I&T process model uses constant test durations, which is sufficient to explain the quantitative decision making process and to compare the current and model-based I&T process for the system upgrade example. In reality, the duration of a test depends on the remaining risk in the system, i.e., on how much risk is already reduced by previous tests, on different risk reduction rates of certain tests, e.g., tests with realizations or with models, and on the stopping criterion of a certain test, e.g., reduce all risk or stop at a certain remaining risk threshold. The industrial application in Section 6.3 uses a more detailed I&T process model that incorporates potential faults and their risks to determine test durations.

Although the I&T process model for the system upgrade does not express differences in durations or costs for testing with realizations or with models, the differences can be determined and analyzed afterwards, based on the generated I&T sequences and on different test costs per time unit for testing with realizations or with models. Taking software qualification testing (category 1) as an example, Table 6.2 shows that the main part of this category can

be executed with realizations only (I&T activity 1Za, 6 time units), while some aspects could possibly be tested with models (I&T activity 1Ma, 2 time units). For the current I&T process, the aspects tested in 1Ma are covered by equivalent realization tests with the same test duration (also 2 time units). Since testing with realizations is usually more expensive than testing with models, we see that the test costs of 8 time units of expensive realization testing in the current I&T process are higher than 6 time units of expensive realization testing and 2 time units of low cost model testing in the MBI&T process. In this way, we can determine and compare the test costs for specific I&T sequences without using different test times in the integration model, which is sufficient for this example.

Based on an I&T process model as described above, the I&T sequencing algorithm from [Boumen 2007] determines all feasible I&T sequences. For the example in this section, an I&T sequence is feasible when all components are integrated via the defined interfaces, when all defined tests are performed, and when the tests are not performed before the required components are integrated. Determining these sequences is based on an assembly-by-disassembly technique [De Mello and Sanderson 1991], which starts with a complete system and iteratively subdivides the system into smaller parts by removing interfaces until only components are left. Reversing the result gives all possible integration sequences from single components to a complete system. Since the number of feasible sequences can be very large, heuristics can be applied to remove sequences that are expected to have a low performance according to the optimization criteria used. For more details on the I&T sequencing algorithm, we refer to [Boumen 2007].

As explained in Chapter 1, the time-to-market T is the most important business driver for ASML, therefore we use the duration of the complete I&T process, the lead time, as the optimization criterion for I&T sequencing. Note that the lead time is different from the total time used for integration and testing, which is the sum of the durations of all separate I&T activities. By performing multiple I&T activities in parallel, the total time used for integration and testing remains equal but the lead time is reduced. To reduce the number of sequences and the corresponding computation time, we applied a heuristic that prefers I&T sequences with as much parallelism as possible.

Figure 6.6 shows the determined I&T sequences for the current (top) and model-based (bottom) I&T process models, in the form of a Microsoft Project Gantt Chart. The figure shows all activities related to component development (dashed bars), component modeling (white bars at the bottom), integration (diamonds), and testing (black bars) over time, and the precedences between the activities (arrows). On the topmost line of the I&T sequences, the long white bar with triangular line ends indicates the lead time.

Several conclusions can be drawn from these sequences. First, the lead time of the MBI&T sequence is shorter, 167 time units against 190 time units for the current I&T sequence, a reduction of 12%. Besides lead time, also the duration of the final system test phase (the long black bars at the right-hand side of Figure 6.6) is important for ASML, since this phase is on the critical path and has a major influence on the time-to-market T. The final system test phase is 78 time units for the MBI&T sequence, 26% less than the 106 time units for the current I&T sequence.

The profitability of using models in the I&T process is determined by quantifying the costs C of enabling the above mentioned benefits of shorter lead time and shorter final sys-

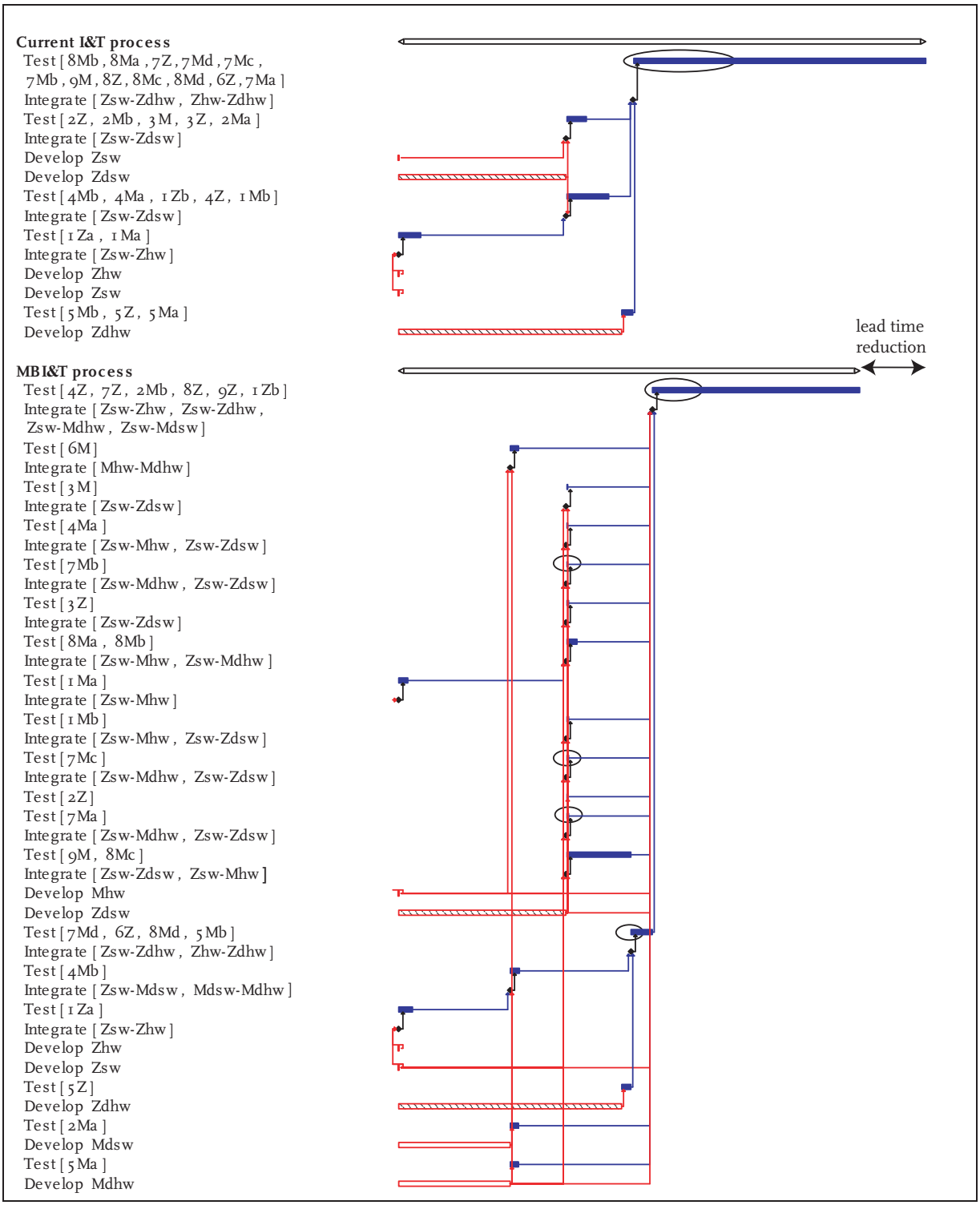


Figure 6.6: Current (top) and model-based (bottom) I&T sequences

tem test phase. As previously mentioned, costs were not explicitly expressed in this system upgrade example but can be compared afterwards, using different costs per time unit for testing with realizations or with models. At ASML, the costs per time unit for realization testing are orders of magnitude higher than for testing software or models in a desktop environment. The use of models influences the costs of the I&T process in both a positive and in a negative sense. On the one hand, the costs of the I&T process increase due to the effort invested in modeling the components, e.g., 80 time units in a relatively low cost desktop environment for the system upgrade example. On the other hand, the costs of the I&T process decrease due to the reduction of time spent on realization testing and due to the possibility to diagnose and fix problems earlier. In Figure 6.6, the current I&T process uses 138 time units of realization testing, while the MBI&T process uses 91 time units of realization testing (34% less) and 47 time units of testing with models, with relatively lower costs. Besides a reduction in these direct costs of testing, the use of models in the I&T process also reduces indirect costs such as diagnosis and fix costs when problems are detected. Compared to the current I&T process, the I&T activities in the MBI&T process are performed earlier and more in parallel, see for example the position of the I&T activities of category 7 in both sequences, indicated by the circles in Figure 6.6. As a result, design and integration problems can be detected and prevented at an earlier stage of development where the costs for fixing them are lower. For example, as explained in Chapter 5, the MBI&T activities applied in the EUV case study prevented problems that, if they would remained undetected, would result in source damage in the real I&T phases, with high costs for fixing the problems and for the accompanying downtime. Although these benefits of early testing cannot directly be expressed in the I&T process model of the system upgrade example, they can be expressed in terms of risk, as shown in the industrial case study in the next section. Summarizing, besides the benefits of a shorter lead time and a shorter final system test phase in the I&T process, using models is also beneficial for the costs of testing, diagnosis, and fixing.

This quantification of benefits and costs can be used as a basis for deciding whether it is profitable to use models in the I&T process. In the system upgrade example, the estimated benefits of using models, e.g., a lead time reduction of 12%, a reduction of realization test time of 34%, and reduced costs for testing, diagnosis, and fixing, probably justify the onetime investments needed for model development, which are 80 time units of modeling in a (low cost) desktop environment.

As an overall result, this example showed the ingredients of the following quantitative decision making process, which can be used to determine when it is profitable to use models in the I&T process, answering QUESTION 2.2:

1. Create an I&T process model with component realizations only.

2. Create a second I&T process model by extending the first I&T process model with component models and by giving a choice of using either a model or a realization for certain I&T activities.

3. Determine the I&T sequences for both I&T process models using the I&T sequencing method.

4. Compare the resulting I&T sequences based on the quantified benefits and costs, e.g., on lead time and test costs.

5. Decide whether the benefits of using models outweigh the additional costs to achieve the benefits.

In the next section, this quantitative decision making process is used in a case study to determine for which parts of a new version of an EUV wafer scanner it is profitable to create a model for early integration and testing.

6.3 Practice: which components of a new EUV wafer scanner should be modeled?

In Chapters 3, 4, and 5, different activities of the MBI&T method were successfully applied to a current version of the EUV wafer scanner. The goal of these applications was to provide a proof of concept showing that models can effectively be used for early integration and testing, answering QUESTION 1.3. Although these chapters showed that the MBI&T method is applicable as well as profitable in industrial practice, the estimated profitability was not a main criterion for deciding where to apply the method for this proof of concept. Instead, these decisions were mainly based on the estimated applicability, e.g., problem size and characteristics, and on the personal involvement of ASML engineers in both the TANGRAM project and in the development of the EUV wafer scanner.

After seeing the applicability and potential profitability of the MBI&T method, the developers of the current version of the EUV wafer scanner would like to know, for future versions of the EUV wafer scanner, which components are the most interesting and profitable candidates for model-based integration and testing. Assuming that a new version of the EUV wafer scanner should be developed, the next subsections describe how the five steps of the quantitative decision making process were executed in order to decide which components should be modeled, providing an answer to QUESTION 2.3, repeated below.

> **QUESTION 2.3** Is it feasible in current industrial practice to quantify the costs of the early I&T method and to decide where and when the method should be applied?

6.3.1 Step 1: Create an I&T process model with component realizations only

In a new version of the EUV wafer scanner, some components of the current version may be reused or adapted, while other components may be new and need to be developed from scratch. These reused, adapted, and new components should then be integrated according to a predetermined system architecture. Figure 6.7 shows an example of a part of such a system architecture, in which the boxes represent the components and the lines represent the interfaces. The figure shows five subsystems, A through E, for which the grey values

of the boxes denote different characteristics of the components. The light grey component realizations (subsystems A and B) are reused from the current version of the EUV wafer scanner, which means that these components will be available earlier and have less risk since they are more mature. In the I&T process model, the development time of a light grey component is 40 time units. The dark grey components (subsystems C, D and E) are newly developed component realizations, which means that they will be available later and have more risk since they have never been used and tested before. In the I&T process model, the development time of a dark grey component is 160 time units. Finally, the white 'switch' components, denoted by *S*, are not part of the real system but they are 'dummy' components to express the risk reduction of testing with models in a more realistic way, which is explained later in this section. Since they are 'dummy' components, they have zero development time and add no risk to the system.

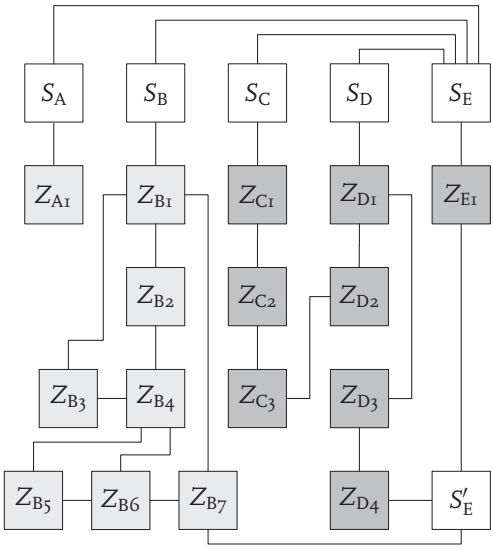


Figure 6.7: Components and interfaces of a new version of the EUV wafer scanner

The interfaces in Figure 6.7 connect the components using different interaction types as explained in Chapter 5. Most interfaces are used to connect components of the same subsystem. Connections between components of different subsystems are mostly established via the interfaces shown at the top of the figure that represent software interfaces, although three other connections exist as well: between Z_{C3} and Z_{D2}, between Z_{B7} and S'_E, and between Z_{D4} and S'_E.

The different characteristics of the components and the interfaces influence the amount of risk that they introduce in the system, as well as the amount of time that is needed to reduce this risk by testing and by fixing the detected problems. In contrast to the system upgrade example, in which Table 6.2 expressed the simple relation between tests, components, and fixed test durations, the I&T process model in this case study incorporates *risk* to express this relation and to determine test durations. In the I&T process model, risk is expressed by multiplying the probability and the impact of *fault states* [Boumen et al. 2008], which denote

possible problems that may be present in the system. Fault states may consider many different types of problems, for instance a broken component, an error in the interface, missing or erroneous functionality, or insufficient performance. A fault state has a certain probability that it is present in the system. When a fault state is present in the system and when it manifests itself in the system behavior, it has a certain impact on the system development process, e.g., restarting the system, replacing a broken component, or even worse, revising the component or system design to prevent the problem. The impact of a fault state usually also depends on the point in time at which its presence is detected: later detection usually means more impact. In the I&T process model of a new version of the EUV wafer scanner, however, differences in fault state impacts are not considered, i.e., impact = 1, which means that the risk only depends on the fault state probability.

Table 6.3 defines the fault states and shows the risk contribution of each component and interface to these fault states. To reduce the size and the complexity of the table, only the components and the interfaces related to subsystems A, B, and E are shown. The components and internal interfaces of subsystem B are grouped and denoted by Z_B* and B-internal*, respectively. On the left-hand side, the table shows five fault states denoted by an f with a subscript: three (internal) fault states for the individual subsystems, one fault state for the interaction between subsystems A and E, and one fault state for the interaction between subsystems B and E. The numbers in the table denote the relative risk contribution of the components and interfaces to each fault state, where different numbers imply different levels of risk contribution, e.g., based on the maturity of a component or on the partitioning of interfaces, which is explained later. For example, Z_{A1} introduces less risk than Z_{E1}, since subsystem A is reused from the current version of the EUV wafer scanner, while subsystem E is newly developed, as indicated by the grey values in Figure 6.7. Note that for Z_B* and B-internal*, each of the grouped components and interfaces has an individual risk contribution of 0.3 to fault state f_B.

Table 6.3: Contribution of components and interfaces to fault states

	components			interfaces							
	Z_{A1}	Z_B*	Z_{E1}	$S_A - Z_{A1}$	$S_A - S_E$	B-internal*	$S_B - Z_{B1}$	$S_B - S_E$	$Z_{B7} - S'_E$	$S_E - Z_{E1}$	$Z_{E1} - S'_E$
f_A	0.3										
f_B		0.3				0.3					
f_E			0.6								
$f_{A\text{-}E}$				0.2	0.6					0.2	
$f_{B\text{-}E}$							0.2	0.6	0.6	0.2	0.2

In a way similar to Table 6.3 that expresses how risk is introduced via components and interfaces, Table 6.4 expresses how risk can be reduced by performing tests. The table shows the fault states f on the left-hand side and possible tests, denoted with t, at the top. The individual subsystems can be tested by t_A, t_B, and t_E. To test the interaction between subsystems, t_{A+E} and t_{B+E} can be used. Finally, the complete system can be tested by t_{bench} using a test

bench with only a part of the system, or by t_{system} using the complete system. The numbers in the table denote the coverage of each test, i.e., the probability that a certain test detects a certain fault state. For example, the internal subsystem fault states are completely covered by the subsystem tests and partially by the interaction tests, which in turn completely cover the interaction fault states. Besides the relation between tests and fault states, the table also shows the duration of each test at the bottom. Different tests may have different durations, e.g., an interaction test takes more time than an internal subsystem test, but less time than a system level test. Note that information about which components need to be realized and integrated to perform a certain test, similar to the second column of Table 6.2, is omitted here, because it is straightforward in this case study: all tests except t_{bench} require all components of the involved subsystems, i.e., either one, two, or all subsystems. The test bench used in t_{bench} consists of all components except for the components on the lowest level of Figure 6.7.

Table 6.4: Tests and their coverage on the fault states

	t_A	t_B	t_E	t_{A+E}	t_{B+E}	t_{bench}	t_{system}
f_A	1			0.5		0.5	1
f_B		1			0.5	0.5	1
f_E			1	0.5	0.5	0.5	1
$f_{A\text{-}E}$				1		0.5	1
$f_{B\text{-}E}$					1	0.5	1
time	8	8	8	16	16	16	24

In this case study, the test durations in the I&T sequences are determined in another way than in the system upgrade example of Section 6.2, which used constant test durations as defined in Table 6.2. In this case study, the test durations are not constant, but they depend on the remaining risk at the moment that a test is executed and thus on the preceding I&T sequence. Here, we only give an informal explanation of how the remaining risk and the expected test duration at a certain point in the I&T sequence are calculated, we refer to [Boumen 2007] for more details. When two system parts, i.e., one component or multiple integrated components, are integrated by connecting an interface between these system parts, the probability of a related fault state, and thus the risk, increases. This increased risk is calculated using the fault state risks of the separate system parts and the risk contribution of the connected interface as defined in Table 6.3. For example, when two system parts with risks of 0.6 and 0.4 for a certain fault state are integrated via an interface that contributes 0.2 risk to that fault state, the increased fault state risk after integration is $1 - (1 - 0.6) * (1 - 0.4) * (1 - 0.2) = 0.808$. When a test is performed on a system part, the risk of a fault state covered by this test decreases. This decreased risk is calculated using the fault state risk before the test and the test coverage as defined in Table 6.4. For example, when a system part has a risk of 0.8 for a certain fault state, and when it passes a test that has 0.5 coverage for that fault state, the risk after testing is $0.8 * (1 - 0.5) = 0.4$. Using risk calculations like these, the fault state risks continuously change with each integration or test activity in the I&T sequence. This influences the expected duration of a particular test

activity in the following way. A test activity may involve the execution of multiple tests, for which the optimal test sequence is determined using a test sequencing algorithm [Boumen et al. 2008]. This algorithm is based on a sequential diagnosis method [Pattipati et al. 1991] and uses AND/OR graphs to represent test trees, showing which tests should be executed depending on whether previous tests passed or failed. The expected duration of such a test tree is calculated by multiplying the test durations defined at the bottom of Table 6.4 with the probabilities that each test in the tree will be executed, which in turn depend on the 'pass' and 'fail' probabilities of previous tests in the tree. The 'pass' and 'fail' probabilities of a test depend on the test coverage as well as on the fault state risks before the test, which are calculated as described above.

6.3.2 Step 2: Extend the I&T process model of step 1 with component models

As shown in Figure 6.7, subsystem E contains a new component E1 that needs to be developed from scratch. Besides that this means relatively long development times as described in the previous subsection, subsystem E also interacts with many other subsystems, which increases the risk of problems. Considering this high risk, it is expected that a model of component E1 would be a good candidate to improve the I&T process of a new version of the EUV wafer scanner. In contrast to component E1, component A1 of subsystem A is reused from the current version of the EUV wafer scanner and it has only one interface, i.e., it is available earlier and has a lower risk. This means that including a model of component A1 probably has a low profitability. By creating I&T process models that include these component models and by analyzing the resulting I&T sequences, we investigate whether the quantitative decision making process can be used to confirm these expectations on the profitability of using a model of components A1 and E1.

Including component models in the I&T process model results in several changes with respect to the information shown in Figure 6.7, Table 6.3 and Table 6.4. As shown in Figure 6.8, additional components (circles) and interfaces are introduced to represent the models M_{A1} and M_{E1}. In this case study, the development time for the models is defined at 16 time units, which is shorter than the 40 and 160 time units of development time for the corresponding realizations Z_{A1} and Z_{E1}. The models have similar interfaces as the corresponding realizations, connecting them to the same 'switch' components, one for M_{A1} and two for M_{E1}. As previously mentioned, these 'switch' components are not part of the real system but they are 'dummy' components to express the risk reduction of testing with models in a more realistic way. Without the 'switch' components in the I&T process model, a model of a component would have interfaces to all other components to which the corresponding realization is connected as well, e.g., M_{E1} in Figure 6.8 would have interfaces to Z_{A1} through Z_{D1}, as well as to Z_{B7} and Z_{D4}. Testing the combination of M_{E1} and some of these other components would then require that the model interfaces, e.g., between M_{E1} and Z_{A1}, are integrated, while the realization interfaces, e.g., between Z_{E1} and Z_{A1}, are not integrated. This means that the risk of the realization interface is not tested with the model, and will completely be introduced when the component realization is integrated via the interface. This does not correspond to reality, since an important aspect of using models for integration and testing is that the

interaction between components, i.e., the risk in the interfaces, is at least partly tested at an early stage. This is the reason why the 'switch' components were introduced and the interfaces were partitioned. For example, a test that uses M_{E1} and Z_{A1} already includes a large part of the interface risk between Z_{E1} and Z_{A1}, namely the interface between S_E and S_A (with a relatively large risk contribution to fault state $f_{A\text{-}E}$, see Table 6.3) and the interface between S_A and Z_{A1}. Together with the interface between S_E and M_{E1}, a large part of the final realization interface and its corresponding risk is reduced when testing with models. Later, when M_{E1} is replaced by Z_{E1}, only the risk related to the interface between the Z_{E1} and S_E is added to the system risk and needs to be reduced by testing.

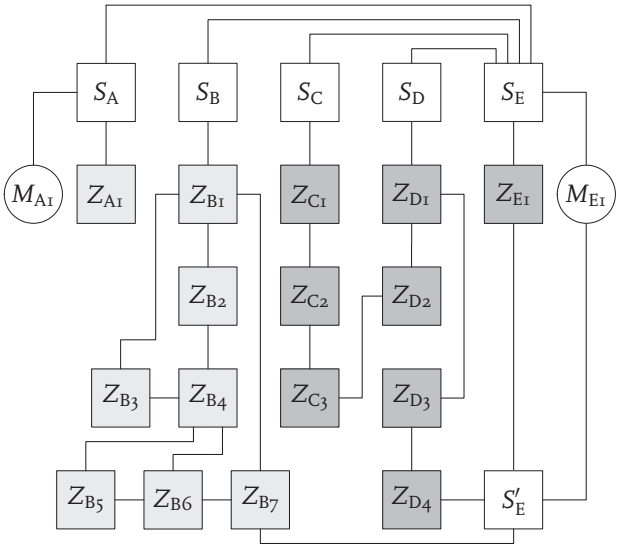


Figure 6.8: Including models of components A1 and E1

Table 6.3 changes in a sense that new columns for the models and the related interfaces are added and their relative risk contribution to each fault state is defined. In principle, models should be abstract representations of the realizations and they should not introduce additional risk on top of the risk introduced by the realizations themselves. In practice, however, there is a possibility that models differ from the actual realization, which introduces some additional risk in the system. This is expressed in Table 6.3 by giving the models and related interfaces a small relative risk contribution to the same fault states as their realization counterparts. For example, M_{E1} gets a relative risk contribution of 0.1 to fault state f_E, and the interface between S_E and M_{E1} gets a relative contribution of 0.1 to both $f_{A\text{-}E}$ and $f_{B\text{-}E}$. This additional risk introduced by the models should also be reduced by testing, which is part of the additional costs of using models in the I&T process. Table 6.4 remains unchanged when models are added. The information about the required components for each test changes in a sense that whenever a test requires Z_{A1} or Z_{E1}, there is also a choice of using the corresponding models M_{A1} and M_{E1}, similar to the choices denoted by the round brackets in Table 6.2. The I&T sequencing algorithm as used in the next step of the quantitative decision making process decides which of the choice alternatives is the most profitable with respect to the used optimization criteria.

6.3.3 Step 3: Determine the I&T sequences for all I&T process models

Applying the changes described in the previous subsection results in three different I&T process models: one without component models (A), one with a model M_{E1} (B), and one with a model M_{A1} (C). For each I&T process model, the (nearly) optimal I&T sequence is determined using the I&T sequencing algorithm from [Boumen 2007], using the lead time as optimization criterion, a 'reduce all risk' stopping criterion for each test activity, and a heuristic that reduces the number of I&T sequences by preferring those that have as much parallelism as possible.

For I&T sequencing, the heuristics should be used with care since their settings regarding the number of considered alternatives combined with the fact that the I&T sequencing algorithm uses estimations to choose between alternatives showed to be quite sensitive for the results. By increasing the number of considered alternatives for a heuristic (with the expectation that better sequences may be found), certain alternatives may be 'overruled' by alternatives that have a higher estimated profitability at the time of making the choice (e.g., shorter lead time by skipping a test), but in fact these alternatives result in a lower profitability when the complete sequence is determined (e.g., since a skipped test leaves more risk that needs be reduced later). To overcome this issue, the I&T sequencing algorithm was executed multiple times with different heuristic settings regarding the number of alternatives taken into consideration. The best results for each I&T process model are shown in Table 6.5, with the lead time (total duration of the I&T sequence), the test time (the amount of time spent on all I&T activities), and the total time (the amount of time spent on all activities, i.e., including development) as well as the relative differences when compared to I&T process model A.

Table 6.5: Results of the three I&T process models

I&T process model	Lead time	Difference	Test time	Difference	Total time	Difference
A. without models	438	-	426	-	1922	-
B. with M_{E1}	396	-9.6%	508	+19.2%	2020	+5.1%
C. with M_{A1}	439	+0.2%	438	+2.8%	1950	+1.5%

Figure 6.9 shows the resulting I&T sequences for I&T process models A and B, in the form of a Microsoft Project Gantt Chart. The figure shows all activities related to development and integration (dashed bars), modeling (white bar at third line from below) and testing (black bars) over time, and the precedences between the activities (arrows). On the topmost line of the I&T sequences, the long white bar with triangular line ends indicates the lead time of the sequence. For simplicity, the development and integration activities of some subsystems and the 'switch' components are grouped (denoted with < and > brackets). Note that the main 'branches' of the two I&T sequences are different. In I&T process model A, one main branch contains subsystems B and D, while the other main branch contains the 'switch' components and the subsystems A, C, and E. In I&T process model B, subsystems C and E are 'moved' from the second main branch to the first main branch with subsystems B and D, while subsystem A and the 'switch' components remain in the second main branch, which now also includes M_{E1} for early testing of the interface between subsystems A and E.

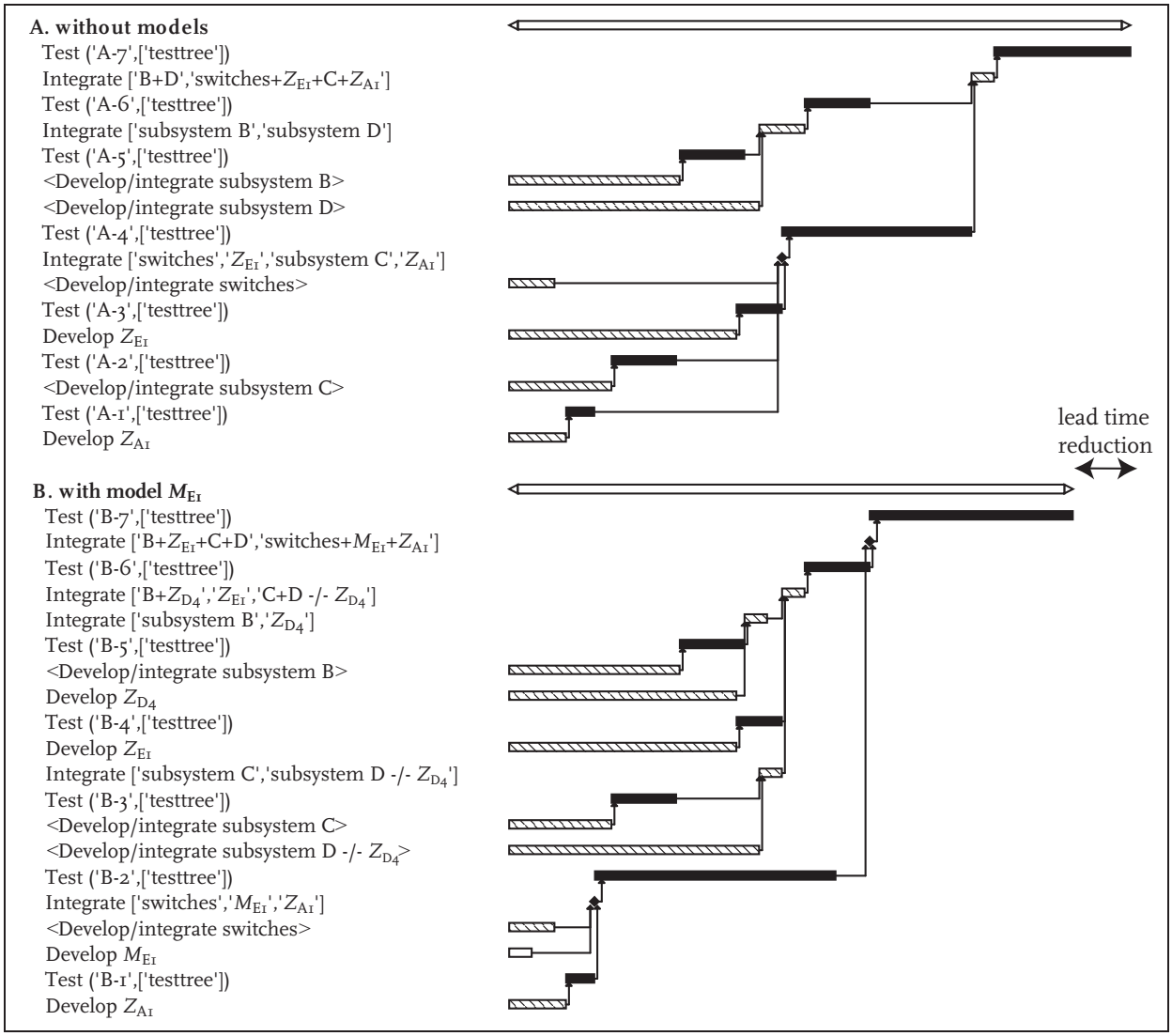
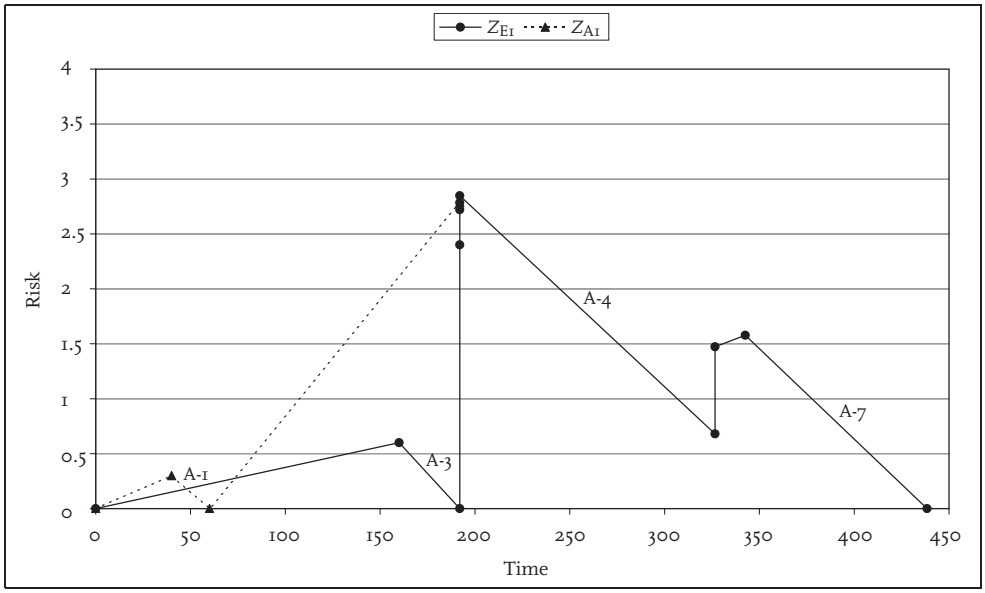
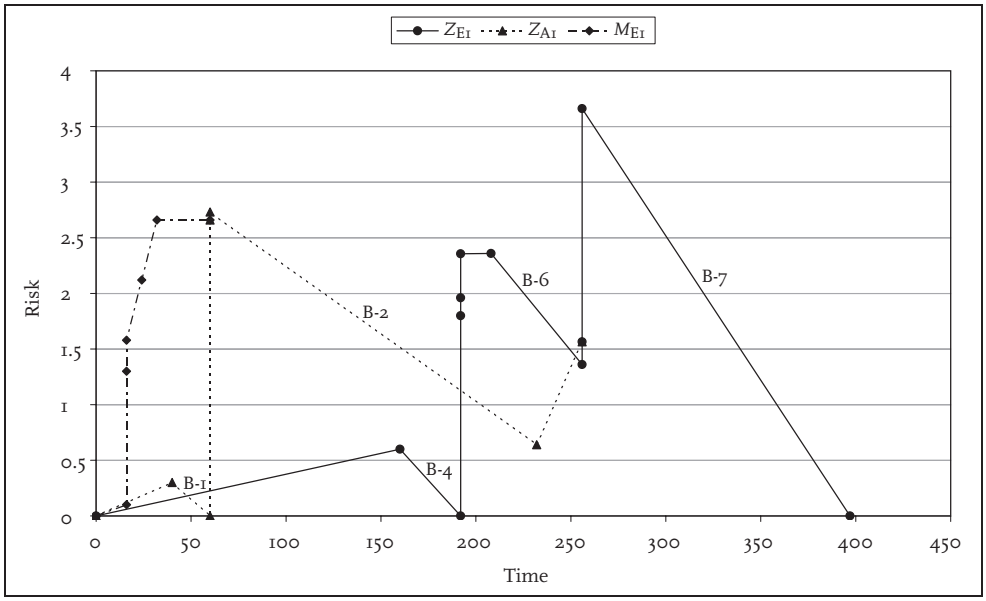


Figure 6.9: I&T sequences for I&T process model A (top) and B (bottom)



(a) I&T process model A (without models)



(b) I&T process model B (with M_{E1})

Figure 6.10: Risk profiles

6.3.4 Step 4: Compare the resulting I&T sequences on quantified benefits and costs

Several conclusions can be drawn from the results shown in Table 6.5 and Figure 6.9. First, these quantitative results support the initial expectations that creating M_{E1} would be profitable, i.e., the lead time is reduced by 9.6%, and that creating M_{A1} would be less profitable, i.e., the lead time even increases with 0.2% due to the additional risk introduced by the model. As expected, M_{E1} enables early testing of the interaction between subsystem E and the other subsystems. For example, by looking at the numbered test trees in Figure 6.9, the interface between subsystems A and E can be tested after 60 time units when M_{E1} is used (test tree B-2), while this is only possible after 192 time units when no models are used (test tree A-4). Second, the test time results in Table 6.5 show that more time is available for testing when a model is used, which increases the system overview and the product quality. This is not only caused by the possibility of earlier testing with the model, but also by the possibility of more parallel testing, as shown by the black testing bars in Figure 6.9. This increased test time is also responsible for the major part of the increased total time as shown in the last column of Table 6.5. The other and much smaller part of the increased total time is caused by model development, which took 16 time units in this case study.

Another view on the effects of using models in the I&T process is shown in Figures 6.10(a) and 6.10(b), which contain the risk profiles for I&T process models A and B, respectively. Similar to the T-Q-C figures of Sections 1.3 and 1.4 in Chapter 1, the risk increases when components are developed and integrated, i.e., potential problems related to the fault states are revealed, and the risk decreases when a system part is tested and the detected problems are fixed (the numbers at the slopes correspond to the test trees in Figure 6.9). For simplicity, the figures only show the risk profiles for Z_{E1}, Z_{A1}, and M_{E1}, because using M_{E1} has the most influence on these risk profiles. Furthermore, linear abstractions of the risk profiles are used since the I&T sequencing algorithm only calculates the risk at the start and at the end of each integration or test activity (the points in the figure); especially the decrease of risk during testing will have a more exponential shape in reality. Also note that when system parts are integrated in the I&T sequence, the corresponding risk profiles are combined and continued as one risk profile. This also explains the higher risk peak at the end of the I&T sequence in Figure 6.10(b), where all components and thus all risk profiles are integrated just before the last test activity (test tree B-7 in Figure 6.9). In Figure 6.10(a), however, the main risk of Z_{E1} and Z_{A1} is tested halfway the I&T sequence (test tree A-4 in Figure 6.9). At this point, subsystems B and D of the other main branch are not integrated yet, i.e., also their risk profiles are not yet combined with the Z_{E1} risk profile in Figure 6.10(a).

The results of the case study are preliminary and need further investigation before real decisions regarding the use of models in the I&T process (step 5) are to be based on it. For example, the relative probabilities and impacts of fault states, as well as the coverage and durations of the tests should be validated, the use of heuristics should be investigated in more detail, and more risk data should be obtained, e.g., using I&T process simulation techniques [De Jong et al. 2007b]. Nevertheless, these preliminary results show that the shapes of the risk profiles in Figure 6.10 correspond to the initial intention of early integration and testing as shown in Figure 1.7 in Chapter 1. By using a model M_{E1}, the system risk is revealed at an

earlier stage, after which it is immediately reduced by testing (in test tree B-2), resulting in a lead time reduction for the I&T process.

6.4 Conclusions

This chapter started with a description of the current I&T process, using a system upgrade example that is common in industry. Nine different categories of I&T activities were identified that cover different system aspects. Since tests can only be performed with realizations, the test costs are relatively high and the tests can only be performed late in the process, where fixing the detected problems is relatively expensive. Subsequently, it was shown how the model-based analysis and testing techniques of the MBI&T method can be applied in each category of I&T activities to enable earlier and more parallel testing with lower costs, which answers QUESTION 2.1.

By using the I&T sequencing method from [Boumen 2007], we showed how (nearly) optimal sequences of I&T activities can be determined and how the costs of using models in the I&T process can be quantified. This quantification of costs supports the decision making process of when the use of models is profitable, thus answering QUESTION 2.2. The results of a basic system upgrade example showed that the lead time and costs of the current I&T process can be reduced by performing tests earlier with models.

Finally, to answer QUESTION 2.3, the proposed quantitative decision making process was applied to the I&T process of a new version of the EUV wafer scanner, showing that it is feasible in current industrial practice to quantify the costs of the MBI&T method and to decide where and when the method should be applied. Three different I&T process models were created that described different scenarios regarding the use of component models. The resulting I&T sequences showed that the profitability of using models can be quantified in terms of reduced lead time, increased and more parallel test time, and risk profiles that show the early revealing of risk which can subsequently be reduced early by testing. The quantitative decision making process supported the initial expectations on the profitability of using different models in the I&T process. Using a model of a new component with many interfaces proved to be profitable (9.4% lead time reduction), while using a model of a reused and more mature component even increased the lead time a bit by 0.2%, due to the additional model risk that has to be reduced by testing as well.

Using this quantitative decision making process in practice requires estimations of the development or delivery times of realizations and models. Furthermore, explicit knowledge is required about possible fault states of the system with their (relative) probability and impact, and about the available tests with their (relative) coverage and duration. As described in [Boumen 2007], ASML test engineers that currently use the I&T sequencing method for periodic software qualification testing (I&T activity 1 of Table 6.1) are able to make these estimations and to maintain the corresponding I&T process model. However, I&T sequencing should not be considered as a 'push the button' technique, since the heuristics and their configuration may be quite sensitive for the results and should thus be used with care.

In this chapter, we focused on quantifying the costs C of reducing the time-to-market T using the MBI&T method. As shown in the previous chapters, reducing time-to-market T is

not the only advantage of the MBI&T method. It can also increase the system overview and the product quality Q, it can test all possible behaviors using model checking, the costs for diagnosing and fixing problems can be reduced, and models enable easier and less expensive testing of exceptional behavior. Although these advantages were not considered in the case study of the quantitative decision making process, they can be expressed in the I&T process model in the following way. As previously mentioned, the effects on product quality can be expressed in terms of risk, e.g., by optimizing the I&T sequences towards minimal risk at a fixed system shipment date. When model checking is considered to be a form of testing the system model with very high coverage, i.e., all possible behaviors, this can be defined in the I&T process model by a high coverage test that can be performed with models only. Although they were not incorporated in the case study, the costs for diagnosing and fixing a fault state can also be expressed in the I&T process model. This can be used to express the lower diagnosis and fix costs when models are used for early testing, e.g., by letting the fix costs increase over time. Furthermore, the tests in the I&T process model can be defined such that they can be performed using only models or only realizations, possibly with different test costs. This allows the modeling of, for example, tests related to machine damage control that can be performed at low costs with models (since machine damage situations can be simulated) but not at all or only at high costs with realizations (with the risk that real machine damage occurs). Finally, 'what if' scenarios can be used to investigate, for example, the effects of developing more detailed models, implying higher model development times, but also a higher coverage of the MBI&T activities and less I&T activities that can be performed with realizations only. Also decisions between longer but low cost model testing and shorter but more expensive realization testing can be made by analyzing 'what if' scenarios. These possible extensions show that I&T sequencing is a suitable technique to get insight in and to analyze possible scenarios for industrial I&T processes, e.g., already during system design, and to make decisions on which I&T activities should be performed in which order, and whether or not models should be used for these I&T activities.

CHAPTER 7

Concluding remarks

In this thesis, a model-based integration and testing (MBI&T) method has been proposed to reduce the disadvantageous effects of the integration and test phases on the time-to-market, product quality, and costs business drivers of high-tech multi-disciplinary system development. The method uses formal and executable models to enable thorough system analysis when no component realizations are available (QUESTION 1.1), as well as automatic component testing and integrated system testing when only some component realizations are available (QUESTION 1.2). Relevant industrial case studies showed that the MBI&T method is applicable and profitable in current industrial practice (QUESTION 1.3). In order to decide where (QUESTION 2.1) and when (QUESTION 2.2) it is profitable to apply this method in an industrial I&T process, a quantitative decision making process has been proposed to quantify and compare the costs of various I&T processes. Also the practical applicability of this quantitative decision making process was shown in an industrial case study (QUESTION 2.3).

In principle, the MBI&T method could be instantiated with paradigms, mathematics, and tools for any system development process in which the components are separately developed and for which it is difficult to analyze and test the system without having a realized and integrated system available. In this thesis, the MBI&T method was instantiated with the concurrent processes paradigm, together with mathematics and tools based on the process algebraic language χ. The hypothesis was that this instantiation fulfills all requirements of the proposed MBI&T activities in Section 2.3 and that it is suitable to perform these activities in industrial practice, focusing on component interaction and time behavior. Chapters 3 through 5 showed that this hypothesis was accepted and that the χ toolset is indeed suitable to apply all MBI&T activities in industrial practice. The χ language is an expressive and compositional language to model systems in the concurrent processes paradigm, including equivalents of other paradigms, e.g., physical phenomena, that are important for the considered system behavior. Due to its formal semantics, the behavior of a χ model can be precisely determined and analyzed, and it can in principle be translated automatically into other forms required for particular analysis and testing techniques such as model checking, automatic model-based testing, and real-time execution in combination with non-χ components.

The main academic contribution of this Ph.D. project is a method based on models in the concurrent processes paradigm that, in addition to simulation and model checking, also provides a consistent and systematic way to integrate and test combinations of models and realizations of industrial systems, using the same modeling language and toolset. Although this model-based integration approach is well-known and common for other paradigms, e.g., hardware-in-the-loop testing in the dynamics and control paradigm, it is a rather new and unexplored area for the concurrent processes paradigm. In the model-based integrated system, the interaction between models and realizations and the real-time behavior are no longer handled according to the formal model semantics. The handling of component interaction and real-time behavior is implemented using a model-based integration infrastructure such that the overall system behavior is equivalent to the modeled and analyzed system behavior, possibly with some acceptable restrictions, e.g., regarding the execution speed. For the industrial examples in Chapter 5, we gave an informal and intuitive indication that the correctness of the infrastructural properties is preserved when the model of the infrastructure is replaced by this model-based integration infrastructure. Formal proofs that these infrastructural properties remain valid for different interaction types and for real-time behavior are left as future work.

The main industrial contribution of this Ph.D. project is that the MBI&T method showed several benefits compared to the current industrial way of working. The modeling activities help to clarify, correct, and complete the design documentation, which directly improves the system quality. By discussing the questions and issues raised in the modeling activities with the involved engineers, their system overview increases and the communication between the engineers improves, which indirectly improves the system quality. Using simulation and model checking techniques, design and integration problems are detected at an earlier stage of system development, where the costs for fixing them are lower. Formal model checking proved to be useful to analyze all possible behaviors of the system, including the exceptional behavior and the non-deterministic behavior which are often insufficiently documented and understood while they may cause major problems and unexpected, hard to diagnose behavior on the system level. Especially for systems with high risks for machine damage or human safety, analyzing all possible behaviors is essential in order to guarantee correct and safe system behavior under all circumstances. Finally, the integration of models and realizations using a model-based integration infrastructure yields an early representation of the real integrated system. The fact that this model-based integrated system can be established much earlier compared to a real integrated system means that integration and testing can start significantly earlier (several months before real integration) and that the total I&T effort is distributed over a wider time frame, i.e., the real I&T phases become less critical. Because of the complete insight in and control over the models used in a model-based integrated system, system tests can be performed more efficiently and with lower costs for diagnosis and fixing, e.g., creating non-nominal conditions to test exceptional behavior is easier, less expensive, and less risky when models are used instead of realizations. Recalling the possible solutions to the I&T problem from Chapter 1, the MBI&T method did not only enable early integration and testing but also contributed to the other two solutions. In the test phase, the risk reduction rate is increased by providing sophisticated and automated analysis and testing techniques with complete insight in and control over the test conditions. In the devel-

opment phase, the maximum system risk is reduced by providing early system level feedback to the component designers. This means that the effects of using the MBI&T method on the T-Q-C business drivers combines the effects as shown in Figures 1.5 through 1.7, enabling a reduction of the time-to-market T and costs C of system development while maintaining or even improving the product quality Q.

Looking at the investigated application area of the MBI&T method, i.e., component interaction and time behavior, the behavior of the encountered systems was not extremely complex, which, of course, is also the intention of the designers. During the MBI&T activities, the level of concurrency and non-determinism remained manageable and comprehensible, e.g., it did not feature true concurrency, and the used interaction types were straightforward. This level of complexity could easily be dealt with using proven technology and tools from academic research, e.g., the computational limits of model checking regarding state space size were not reached, and, for model-based integration, the asynchronous interaction types used in the application could easily be modeled and implemented using additional buffer processes.

One of the lessons learned in this Ph.D. project is that, when applying academic results in industry, one should realize and accept that the (not always optimal) conditions and constraints, e.g., incomplete and ambiguous designs, are part of the current industrial problem which should be and can be dealt with, also when using formal techniques that often rely on complete specifications.

Another lesson learned is that when models are used for integration and testing, the perspective on the goal of modeling is different than for model-based approaches used in system development. Many model-based approaches are 'top-down', i.e., the *intended* system behavior is modeled, after which correctness preserving model transformations are used to obtain more detailed models (refinement) and eventually component realizations (code generation). In contrast to this, models for integration and testing should represent the *actual* component and system behavior, i.e., as it is designed, including the issues and potential system level problems. Then, using a 'bottom-up' approach, the 'as is' component models are integrated and the (potentially incorrect) emerging system behavior, e.g., the interaction and time behavior of the components, is determined such that system level problems are detected as early as possible.

As intended, the activities of the MBI&T method are complementary to the current way of working and do not require changes in the system development process. Creating the models is based on the 'as is' component designs, independent of the form of these designs (mental model, document, computer model) and without posing additional requirements on the current way of working for the requirements definition, design, and realization phases. However, the activities do require a certain amount of time to be invested by the involved engineers for answering questions about unclarities in their designs and for discussing the models and the analysis results. In return, the MBI&T activities provide them with valuable feedback on the system behavior and increase their system overview. In addition to these time investments, a MBI&T activity that involves component realizations requires that test resources, e.g., a component realization or a prototype system, are allocated to perform the MBI&T activity. This may influence the current I&T process in a sense that there are more possible I&T activities to which the (limited) test resources must be allocated. In order to

determine whether these additional costs introduced by the MBI&T method outweigh the benefits, and to determine when it is profitable to use the MBI&T method, the quantitative decision making process as proposed in Chapter 6 can be used.

Looking back at this Ph.D. project, it is clear that it is a good example of the 'industry as laboratory' approach utilized by the Embedded Systems Institute: proven technology and tools from academic research were used, adapted, and combined in order to apply them in industrial practice. The proof of concept application focused on a real industrial system in its development phase. As a consequence and as a side-effect, some immediate lessons were learned by the involved engineers which helped them in their daily work and enabled them to improve the system quality at an earlier stage. In this manner, it has been proven in theory and in practice that the MBI&T method really reduces the I&T lead time and thus time-to-market and costs for ASML. Furthermore, this project increased the awareness of ASML engineers that models offer a systematic way of working, which enables them to obtain more complete and less ambiguous designs, and, with proper model semantics, to automatically analyze all possible behaviors of a system, which is essential but hardly possible in the current, document-based way of working.

References

Airbus (2007). Website. http://www.airbus.com.

American Heritage Dictionary of the English Language (2007). Definition of "paradigm" at dictionary.com. http://dictionary.reference.com/browse/paradigm.

ASML (2007). Website. http://www.asml.com.

Awerbuch, B. (1985). Complexity of network synchronization. *Journal of the ACM*, vol. 32, no. 4, pp. 804–823.

Baeten, J.C.M. and Weijland, W.P. (1990). *Process algebra*, vol. 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

Baeten, J.C.M. and Bergstra, J.A. (1992). Asynchronous communication in real space process algebra. In: *Proceedings of the 2nd International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'92), Nijmegen, the Netherlands*, vol. 571 of *Lecture Notes in Computer Science*, pp. 473–492. Springer-Verlag.

Baeten, J.C.M., van Beek, D.A., and Rooda, J.E. (2007). Process algebra. In: *CRC Handbook of Dynamic System Modeling*, chap. 19, pp. 19.1–19.19. Chapman & Hall/CRC.

von der Beeck, M., Braun, P., Rappl, M., and Schroder, C. (2002). Model-based requirements engineering for embedded software. In: *Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'02), Essen, Germany*, p. 92.

van Beek, D.A. and Rooda, J.E. (2000). Languages and applications in hybrid modelling and simulation: positioning of Chi. *Control Engineering Practice*, vol. 8, no. 1, pp. 81–91.

van Beek, D.A., van der Ham, A., and Rooda, J.E. (2002). Modelling and control of process industry batch production systems. In: *Proceedings of the 15th Triennial World Congress of the International Federation of Automatic Control (IFAC'02), Barcelona, Spain*. CD-ROM.

van Beek, D.A., Man, K.L., Reniers, M.A., Rooda, J.E., and Schiffelers, R.R.H. (2005). Syntax and semantics of timed Chi. Computer Science report 05–09, Eindhoven University of Technology.

van Beek, D.A., Man, K.L., Reniers, M.A., Rooda, J.E., and Schiffelers, R.R.H. (2006a). Syntax and consistent equation semantics of hybrid Chi. *Journal of Logic and Algebraic Programming – Special issue on Process Theory for Hybrid Systems*, vol. 68, no. 1–2, pp. 129–210.

van Beek, D.A., Man, K.L., Reniers, M.A., Rooda, J.E., and Schiffelers, R.R.H. (2006b). Deriving simulators for hybrid chi models. In: *Proceedings of IEEE International Symposium on Computer-Aided Control Systems Design (CACSD'06), Munich, Germany*, pp. 42–49.

van Beek, D.A., Hofkamp, A.T., Reniers, M.A., Rooda, J.E., and Schiffelers, R.R.H. (2008). Syntax and formal semantics of Chi 2.0. Systems Engineering report 2008–01, Eindhoven University of Technology. ISSN: 1872–1567.

Behrmann, G., Hune, T., and Vaandrager, F.W. (2000). Distributing timed model checking – how the search order matters. In: *Proceedings of 12th International Conference on Computer Aided Verification (CAV'00), Chicago, IL, USA*, vol. 1855 of *Lecture Notes in Computer Science*, pp. 216–231. Springer-Verlag.

Behrmann, G., David, A., Håkansson, J., Hendriks, M., Larsen, K.G., Pettersson, P., and Yi, W. (2006). UPPAAL 4.0. In: *Proceedings of the 3rd International Conference on Quantitative Evaluation of Systems (QEST'06), Riverside, CA, USA*, pp. 125–126.

Bengtsson, J. and Yi, W. (2004). Timed automata: semantics, algorithms and tools. In: *Lectures on Concurrency and Petri Nets*, vol. 3098 of *Lecture Notes in Computer Science*, pp. 87–124. Springer-Verlag.

Benz, S. (2007). Combining test case generation for component and integration testing. In: *Proceedings of the 3rd international workshop on Advances in Model-Based Testing (A-MOST'07), London, UK*, pp. 23–33. ACM Press.

van der Bijl, M., Rensink, A., and Tretmans, J. (2003). Compositional testing with *ioco*. In: *Proceedings of the 3rd International Workshop on Formal Approaches to Software Testing (FATES'03), Montreal, Canada*, vol. 2931 of *Lecture Notes in Computer Science*, pp. 86–100. Springer-Verlag.

Boehm, B.W. (1981). *Software engineering economics*. Prentice Hall.

Boehm, B.W. and Basili, V.R. (2001). Software defect reduction top 10 list. *IEEE Computer*, vol. 34, no. 1, pp. 135–137.

de Boer, F.S., Klop, J.W., and Palamidessi, C. (1992). Asynchronous communication in process algebra. In: *Proceedings of the 7th annual IEEE symposium on Logics in Computer Science (LICS'92), Los Alamitos, CA, USA*, pp. 137–147. IEEE Computer Society.

Bohnenkamp, H. and Belinfante, A. (2005). Timed testing with TORX. In: *Proceedings of the International Symposium of Formal Methods Europe (FM'05), Newcastle, UK*, vol. 3582 of *Lecture Notes in Computer Science*, pp. 173–188. Springer-Verlag.

Bortnik, E.M., van Beek, D.A., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2005a). Verification of timed Chi models using UPPAAL. In: *Proceedings of the 2nd International Conference on Informatics in Control, Automation and Robotics (ICINCO'05), Barcelona, Spain*, pp. 486–492. INSTICC Press.

Bortnik, E.M., Trčka, N., Wijs, A.J., Luttik, S.P., van de Mortel-Fronczak, J.M., Baeten, J.C.M., Fokkink, W.J., and Rooda, J.E. (2005b). Analyzing a χ model of a turntable system using SPIN, CADP and UPPAAL. *Journal of Logic and Algebraic Programming*, vol. 65, no. 2, pp. 51–104.

Bortnik, E.M., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2007). Verifying Chi models in UPPAAL. Systems Engineering report 2007–06, Eindhoven University of Technology. ISSN: 1872–1567.

Boumen, R., de Jong, I.S.M., Vermunt, J.W.H., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2006). A risk-based stopping criterion for test sequencing. Internal Report SE 420460, Eindhoven University of Technology. Submitted for publication in IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans.

Boumen, R. (2007). *Integration and test plans for complex manufacturing systems*. Ph.D. thesis, Eindhoven University of Technology.

Boumen, R., de Jong, I.S.M., Vermunt, J.W.H., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2008). Test sequencing in complex manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, vol. 38, no. 1, pp. 25–37.

Braspenning, N.C.W.M., Kostić, D., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2006a). Model-based support for early integration and testing of a multi-disciplinary industrial system. In: *Proceedings of the 5th European Systems Engineering Conference (EuSEC'06), Edinburgh, UK*. CD-ROM.

Braspenning, N.C.W.M., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2006b). A model-based integration and testing method to reduce system development effort. *Electronic Notes in Theoretical Computer Science – Proceedings of the 2nd workshop on Model-Based Testing (MBT'06), Vienna, Austria*, vol. 164, no. 4, pp. 13–28.

Braspenning, N.C.W.M., Boumen, R., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2007a). A quantitative method to decide where and when it is profitable to use models for integration and testing. Systems Engineering report 2007–14, Eindhoven University of Technology. ISSN: 1872–1567. Submitted for publication in Computers in Industry.

Braspenning, N.C.W.M., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2007b). Analysis and implementation of infrastructure for model-based integration and testing. In: *Proceedings of the 5th Annual Conference on Systems Engineering Research (CSER'07), Hoboken, NJ, USA*. CD-ROM. Available online at http://www.stevens.edu/cser/.

Braspenning, N.C.W.M., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2007c). Modeling, analysis, and implementation of infrastructure for model-based integration and testing. Systems Engineering report 2007–08, Eindhoven University of Technology. ISSN: 1872–1567. Submitted for publication in IEEE Transactions on Automation Science and Engineering.

Braspenning, N.C.W.M., van der Ploeg, D.O., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2007d). Model-based techniques for intelligent integration and testing in industry. In: *Proceedings of the 17th International Symposium of INCOSE (INCOSE'07), San Diego, CA, USA*. CD-ROM.

Braspenning, N.C.W.M., Bortnik, E.M., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2008). Model-based system analysis using Chi and UPPAAL: an industrial case study. *Computers in Industry*, vol. 59, no. 1, pp. 41–54.

Bratthall, L.G., Runeson, P., Ädelsward, K., and Eriksson, W. (2000). A survey of lead-time challenges in the development and evolution of distributed real-time systems. *Information and Software Technology*, vol. 42, no. 13, pp. 947–958.

Brinksma, E. and Tretmans, J. (2001). Testing transition systems: an annotated bibliography. In: *Revised tutorial lectures of the 4th Summer School on Modelling and Verification of Parallel Processes (MOVEP'00), Nantes, France*, vol. 2067 of *Lecture Notes in Computer Science*, pp. 187–195. Springer-Verlag.

Brooks, F.P. (1995). *The mythical man-month: essays on software engineering – 20th anniversary edition*. Addison-Wesley Professional, 1st edn.

Brugman, T. and Beenker, F. (2003). TANGRAM project plan. Technical report 2002–10060, ASML and Embedded Systems Institute (ESI).

Chandrupatla, T.R. and Belegundu, A.D. (2002). *Introduction to finite elements in engineering*. Prentice Hall, 3rd edn.

Curtis, B., Krasner, H., and Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM*, vol. 31, no. 11, pp. 1268–1287.

David, A., Behrmann, G., Larsen, K.G., and Yi, W. (2003). A tool architecture for the next generation of UPPAAL. In: *UNU/IIST 10th Anniversary Colloquium – Formal Methods at the Cross Roads: from Panacea to Foundational Support, Lisbon, Portugal*, no. 2757 in Lecture Notes in Computer Science, pp. 352–366. Springer-Verlag.

Demaine, E.D. (1998). Protocols for non-deterministic communication over synchronous channels. In: *Proceedings of the 12th International Parallel Processing Symposium and 9th Symposium on Parallel and Distributed Processing (IPPS–SPDP'98), Orlando, FL, USA*, pp. 24–30. IEEE Computer Society.

Denissen, W.J.A. (2006). A multidisciplinary model-based test and integration infrastructure. In: *Proceedings of IEEE International Symposium on Computer-Aided Control Systems Design (CACSD'06), Munich, Germany*, pp. 1916–1921.

Deppe, M., Zanella, M., Robrecht, M., and Hardt, W. (2004). Rapid prototyping of real-time control laws for complex mechatronic systems: a case study. *Journal of Systems and Software*, vol. 40, no. 3, pp. 263–274.

Diethers, K. and Huhn, M. (2004). Vooduu: verification of object-oriented designs using UPPAAL. In: *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04), Barcelona, Spain*, vol. 2988 of *Lecture Notes in Computer Science*, pp. 139–143. Springer-Verlag.

Embedded Systems Institute (2006). ESI research agenda on embedded systems engineering. Tech. rep., Embedded Systems Institute.

Embedded Systems Institute (2007). Website. http://www.esi.nl.

Engel, E., Bogomolni, I., Shachar, S., and Grinman, A. (2004). Gathering historical lifecycle quality costs to support optimizing the VVT process. In: *Proceedings of the 14th International Symposium of INCOSE (INCOSE'04), Toulouse, France*. CD-ROM.

Eugster, P.Th., Felber, P.A., Guerraoui, R., and Kermarrec, A. (2003). The many faces of publish/subscribe. *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131.

FEI Company (2007). Website. http://www.fei.com.

Fewster, M. and Graham, D. (1999). *Software test automation: effective use of test execution tools*. Addison Wesley.

Fleurkens, J.W.G. (1996). *Interactive modelling and simulation of heterogeneous systems*. Ph.D. thesis, Eindhoven University of Technology.

Fokkink, W.J. (2000). *Introduction to process algebra*. Texts in Theoretical Computer Science, An EATCS Series. Springer-Verlag, 1st edn.

Franklin, G.F., Powell, J.D., and Emami-Naeini, A. (2005). *Feedback control of dynamic systems*. Prentice Hall, 5th edn.

Frantzen, L., Tretmans, J., and Willemse, T.A.C. (2006). A symbolic framework for model-based testing. In: *Proceedings of Formal Approaches to Testing and Runtime Verification (FATES/RV'06), Seattle, WA, USA*, vol. 4262 of *Lecture Notes in Computer Science*, pp. 40–54. Springer-Verlag.

Freriks, H.J.M., Heemels, W.P.M.H., Muller, G.J., and Sandee, J.H. (2006). On the systematic use of budget-based design. In: *Proceedings of the 16th International Symposium of INCOSE (INCOSE'06), Orlando, FL, USA*. CD-ROM.

Fuchs, N.E. (1992). Specifications are (preferably) executable. *IEE/BCS Software Engineering Journal*, vol. 7, no. 5, pp. 323–334.

Gomaa, H. (2000). *Designing concurrent, distributed, and real-time applications with UML*. Addison-Wesley Professional, 1st edn.

Groote, J.F. (1988). Implementations of events in LOTOS specifications. Technical report 009/88EN, Philips CFT, Eindhoven, the Netherlands.

Haagh, P.A.M., Wilkens, A.U., Rulkens, H.J.A., van Campen, E.J.J., and Rooda, J.E. (1998). Application of a layout design method to the dielectric decomposition area in a 300 mm wafer fab. In: *Proceedings of the 7th International Symposium on Semiconductor Manufacturing (ISSM'98), Tokyo, Japan*, pp. 69–72. Ultra Clean Society.

Halbwachs, N. and Baghdadi, S. (2002). Synchronous modeling of asynchronous systems. In: *Proceedings of the 2nd International Workshop on Embedded Software (EMSOFT'02), Grenoble, France*, vol. 2491 of *Lecture Notes in Computer Science*, pp. 240–251. Springer-Verlag.

Hanselmann, H. (1996). Hardware-in-the-loop simulation testing and its integration into a CACSD toolset. In: *Proceedings of the 1996 IEEE International Symposium on Computer-Aided Control System Design (CACSD'96), Dearborn, MI, USA*, pp. 152–156.

Hartman, A. (2002). Model-based test generation tools. AGEDIS report, AGEDIS project.

Herbsleb, J.D. and Kuwana, E. (1993). Preserving knowledge in design projects: what designers need to know. In: *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '93), Amsterdam, the Netherlands*, pp. 7–14. ACM Press.

Hofkamp, A.T. and Rooda, J.E. (2007). Chi 1.0 reference manual. Tech. rep., Eindhoven University of Technology. Available online at http://se.wtb.tue.nl/sewiki/chi.

Holzmann, G.J. (1997). The model checker SPIN. *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295.

Hooman, J., Mulyar, N., and Posta, L. (2004). Coupling SIMULINK and UML models. In: *Proceedings of Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMATS'04), Braunschweig, Germany*, pp. 304–311.

Horch, J.W. (2003). *Practical guide to software quality management*. Artech House, 2nd edn.

Huang, J., Voeten, J., and Corporaal, H. (2006). Correctness-preserving synthesis for real-time control software. In: *Proceedings of the 6th International Conference on Quality Software (QSIC'06), Beijing, China*, pp. 65–73. IEEE Computer Society.

Hull, E., Jackson, K., and Dick, J. (2005). *Requirements engineering*. Springer, 2nd edn.

Hurwitz, J. (1998). Sorting out middleware. *DBMS Magazine*, vol. 11, no. 1, pp. 10–12.

INCOSE (2006). *Systems engineering handbook*. INCOSE, 3rd edn.

JABBER Software Foundation (2007). Website. http://www.jabber.org/.

Jones, G. (2001). Programming in occam – web edition. Available online at http://web.comlab.ox.ac.uk/oucl/work/geraint.jones/publications/book/Pio1/. Originally in: Prentice Hall International Series in Computer Science, 1988.

de Jong, I.S.M., Boumen, R., van de Mortel-Fronczak, J.M, and Rooda, J.E. (2006). Integration and test strategies for semi-conductor manufacturing equipment. In: *Proceedings of the 16th International Symposium of INCOSE (INCOSE'06), Orlando, FL, USA*. CD-ROM.

de Jong, I.S.M., Boumen, R., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2007a). An overview of integration and test plans in organizations with different business drivers. In: *Proceedings of the 5th Annual Conference on Systems Engineering Research (CSER'07), Hoboken, NJ, USA*. CD-ROM. Available online at http://www.stevens.edu/cser/.

de Jong, I.S.M., Boumen, R., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2007b). Test strategy analysis for manufacturing systems. Systems Engineering report 2007–10, Eindhoven University of Technology. ISSN: 1872–1567. Submitted for publication in IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans.

Katoen, J-P. (1999). *Concepts, algorithms and tools for model checking*, vol. 32–1 of *Arbeitsberichte der Informatik*. Friedrich-Alexander-Universität Erlangen-Nürnberg.

Kleppe, A., Bast, W., and Warmer, J. (2003). *MDA explained – the model driven architecture: practice and promise*. Addison-Wesley Professional, 1st edn.

Knabe, F. (1993). A distributed protocol for channel-based communication with choice. *Computers and Artificial Intelligence*, vol. 12, no. 5, pp. 475–490.

Lindahl, M., Pettersson, P., and Yi, W. (2001). Formal design and analysis of a gearbox controller. *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 3, no. 3, pp. 353–368.

Liu, J.S. (2001). *Monte Carlo strategies in scientific computing*. Springer Series in Statistics. Springer-Verlag.

Man, K.L. and Schiffelers, R.R.H. (2006). *Formal specification and analysis of hybrid systems*. Ph.D. thesis, Eindhoven University of Technology.

Martin, J.N. (1996). *Systems engineering guidebook: a process for developing systems and products*. CRC Press.

de Mello, L.S.H. and Sanderson, A.C. (1991). A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Transactions on Robotics and Automation*, vol. 7, no. 2, pp. 228–240.

Millard, P., Saint-Andre, P., and Meijer, R. (2006). XEP–0060: publish-subscribe. Jabber Software Foundation. Available online at http://www.xmpp.org/extensions/xep-0060.html.

Milner, R. (1989). *Communication and concurrency*. Prentice Hall.

MODELICA project (2007). Website. http://www.modelica.org/.

Mörk, Simon (2001). Distributed implementation of a process algebra based programming language for embedded systems. *Nordic Journal of Computing*, vol. 8, no. 1, pp. 121–158.

van de Mortel-Fronczak, J.M., Vervoort, J., and Rooda, J.E. (2001). Simulation-based design of machine control systems. In: *Proceedings of the 15th European Simulation Multiconference, Prague, Czech Republic*.

Mousavi, M.R., le Guernic, P., Talpin, J-P., Shukla, S.K., and Basten, T. (2004). Modeling and validating globally asynchronous design in synchronous frameworks. In: *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'04), Paris, France*, pp. 384–389. IEEE Computer Society.

Muller, G. (2007). Coping with system integration challenges in large complex environments. In: *Proceedings of the 17th International Symposium of INCOSE (INCOSE'07), San Diego, CA, USA*. CD-ROM.

NASA (2007). Website. http://www.nasa.gov.

National Instruments (2007). Compact Fieldpoint product information. Website. http://www.ni.com/compactfieldpoint.

Nestmann, U. (2000). What is a 'good' encoding of guarded choice? *Journal of Information and Computation*, vol. 156, no. 1–2, pp. 287–319.

Océ (2007). Website. http://www.oce.com.

Ogren, I. (2000). On principles for model-based systems engineering. *Systems Engineering*, vol. 3, no. 1, pp. 38–49.

Opto 22 (2007). SNAP PAC System product information. Website. http://www.opto22.com/ad/pac.aspx.

van Osch, M.P.W.J. (2006). Hybrid input-output conformance and test generation. In: *Proceedings of Formal Approaches to Testing and Runtime Verification (FATES/RV'06), Seattle, WA, USA*, vol. 4262 of *Lecture Notes in Computer Science*, pp. 70–84. Springer-Verlag.

van Osch, M.P.W.J. (2007). Model-based testing of hybrid systems. In: J. Tretmans, editor, TANGRAM*: model-based integration and testing of complex high-tech systems*, chap. 10, pp. 129–141. Embedded Systems Institute, Eindhoven, the Netherlands. See also [Tretmans 2007].

Palamidessi, C. (1997). Comparing the expressive power of the synchronous and the asynchronous π-calculus. In: *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'97), Paris, France*, pp. 256–265. ACM Press.

Pattipati, K.R., Deb, S., Dontamsetty, M., and Maitra, A. (1991). START: system testability analysis and research tool. *IEEE Aerospace and Electronic Systems Magazine*, vol. 6, no. 1, pp. 13–20.

Philips Medical Systems (2007). Website. http://www.medical.philips.com.

Pietersma, J. and van Gemund, A.J.C. (2007). Benefits and costs of model-based fault diagnosis for semiconductor manufacturing equipment. In: *Proceedings of the 17th International Symposium of INCOSE (INCOSE'07), San Diego, CA, USA*. CD-ROM.

Potts, C. (1993). Software-engineering research revisited. *IEEE Software*, vol. 10, no. 5, pp. 19–28.

Prenninger, W. and Pretschner, A. (2004). Abstractions for model-based testing. *Electronic Notes in Theoretical Computer Science – Proceedings of the International Workshop on Test and Analysis of Component Based Systems (TACoS'04), Barcelona, Spain*, vol. 116, pp. 59–71.

Prins, M. (2004). Testing industrial embedded systems – an overview. In: *Proceedings of the 14th International Symposium of INCOSE (INCOSE'04), Toulouse, France*. CD-ROM.

PTOLEMY project (2007). Website. http://ptolemy.eecs.berkeley.edu/.

Requirements Assistant (2007). Website. http://www.requirementsassistant.nl.

Rook, P. (1986). Controlling software projects. *Software Engineering Journal*, vol. 1, no. 1, pp. 17–16.

Rowson, J.A. (1994). Hardware/software co-simulation. In: *Proceedings of the 31st Design Automation Conference (DAC'94), San Diego, CA, USA*, pp. 439–440. ACM Press.

Saleh, R.A. and Newton, R.A. (1990). *Mixed-mode simulation*. Kluwer Academic Publishers.

Sorqvist, L. (1998). *Poor quality costing*. Ph.D. thesis, Royal Institute of Technology, Stockholm, Sweden.

SPICE (2007). Website. http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/.

SPIN (2007). Website. http://spinroot.com.

SRON (2007). Website. http://www.sron.nl.

SYSML (2007). Website. http://www.sysml.org.

Systems Engineering Group (2007). Chi language and tools. Mechanical Engineering Department, Eindhoven University of Technology, Website. http://se.wtb.tue.nl/sewiki/chi.

TANGRAM project (2007). Website. http://www.esi.nl/tangram.

The Mathworks – SIMULINK (2007). Website. http://www.mathworks.com/products/simulink/.

The Mathworks – SIMULINK design verifier (2007). Website. http://www.mathworks.com/products/sldesignverifier/.

The Mathworks – STATEFLOW (2007). Website. http://www.mathworks.com/products/stateflow/.

Tijms, H.C. (2003). *A first course in stochastic models*. Wiley.

TorX (2007). Website. http://www.purl.org/net/torx/.

Tretmans, J. (1996). Test generation with inputs, outputs and repetitive quiescence. *Software–Concepts and Tools*, vol. 17, no. 3, pp. 103–120.

Tretmans, J., editor (2007). Tangram*: model-based integration and testing of complex high-tech systems*. Embedded Systems Institute, Eindhoven, the Netherlands. Available online at http://www.esi.nl/tangram/.

Trčka, N. (2006). Verifying Chi models of industrial systems with Spin. In: *Proceedings of the 8th International Conference on Formal Engineering Methods (ICFEM'06), Macao, China*, vol. 4260 of *Lecture Notes in Computer Science*, pp. 132–148. Springer-Verlag.

Twisted Matrix Labs (2007). Website. http://twistedmatrix.com.

UML (2007). Website. http://www.uml.org.

Uppaal (2007). Website. http://www.uppaal.com.

Verhoef, M., Visser, P., Hooman, J., and Broenink, J. (2007). Co-simulation of distributed embedded real-time control systems. In: *Proceedings of 6th International Conference on Integrated Formal Methods (IFM'07), Oxford, UK*, vol. 4591 of *Lecture Notes in Computer Science*, pp. 639–658. Springer-Verlag.

de Vries, R.G. and Tretmans, J. (2000). On-the-fly conformance testing using Spin. *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 2, no. 4, pp. 382–393.

de Vries, R.G. and Tretmans, J. (2001). Towards formal test purposes. In: *Proceedings of the Workshop on Formal Approaches to Testing of Software (FATES'01), Aalborg, Denmark*, vol. NS–01–4 of *BRICS Notes Series*, pp. 61–76. University of Aarhus.

Willemse, T.A.C. (2006). Heuristics for *ioco*-based test-based modelling. In: *Proceedings of the 11th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'06), Bonn, Germany*, vol. 4346 of *Lecture Notes in Computer Science*, pp. 132–147. Springer-Verlag.

Williams, D. and Ambler, A.P. (2002). System manufacturing test cost model. In: *Proceedings of the IEEE International Test Conference (ITC'02), Baltimore, MD, USA*, pp. 482–490.

de Wulf, M., Doyen, L., and Raskin, J-F. (2005). Almost ASAP semantics: from timed models to timed implementations. *Formal Aspects of Computing*, vol. 17, no. 3, pp. 319–341.

Wünsche, S. (1996). Simulator coupling for electro-thermal simulation of integrated circuits. In: *Proceedings of the 2nd International Workshop on Thermal Investigations of IC's and Microstructures (Therminic'96), Budapest, Hungary*, pp. 89–93.

Curriculum Vitae

N.C.W.M. (Niels) Braspenning was born on the 18th of June, 1979 in Breda, the Netherlands. In 1997, he finished VWO at the Katholieke Scholengemeenschap Etten-Leur in Etten-Leur, the Netherlands. From 1997 to 2003, he studied Mechanical Engineering at the Eindhoven University of Technology, the Netherlands. Within the Systems Engineering Group, he performed his graduation project on the topic 'Scheduling and Behavior Verification of Machines based on Task-Resource Models', as part of a Ph.D. project performed at ASML. After graduating in 2003, he started his Ph.D. project within the same group, on the topic 'Model-based Integration and Testing of High-tech Multi-disciplinary Systems'. This Ph.D. project is part of the TANGRAM project on integration and testing, in close co-operation with ASML, the Embedded Systems Institute, and other industrial and academic partners.

Titles in the IPA Dissertation Series since 2002

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing*. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in µCRL*. Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand*. Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks*. Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions*. Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components*. Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components*. Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method*. Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving*. Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems*. Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation*. Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments*. Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications*. Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems*. Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell*. Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation*. Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets*. Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics*. Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station*. Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents*. Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures*. Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling*. Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System*

Architectures. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability*. Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions*. Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments*. Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages*. Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics*. Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Faculty of Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication*. Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell*. Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems*. Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality*. Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting*. Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs*. Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations*. Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data*. Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space*. Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization*. Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavicius. *Hybrid Techniques for Hybrid Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

M.E. Warnier. *Language Based Security for Java and JML*. Faculty of Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. *At Home In Service Discovery*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. *Expressivity of Timed Automata Models*. Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. *Formalising Interface Specifications*. Faculty of Mathematics and Computer Science, TU/e. 2006-19

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Faculty of Mathematics and Computer Science, TU/e. 2006-20

J.V. Guillen Scholten. *Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition*. Faculty of Mathematics and Natural Sciences, UL. 2006-21

H.A. de Jong. *Flexible Heterogeneous Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems*. Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2007-06

M.W.A. Streppel. *Multifunctional Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2007-07

N. Trčka. *Silent Steps in Transition Systems and Markov Chains*. Faculty of Mathematics and Computer Science, TU/e. 2007-08

R. Brinkman. *Searching in encrypted data*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09

A. van Weelden. *Putting types to good use*. Faculty of Science, Mathematics and Computer Science, RU. 2007-10

J.A.R. Noppen. *Imperfect Information in Software Development Processes*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-11

R. Boumen. *Integration and Test plans for Complex Manufacturing Systems*. Faculty of Mechanical Engineering, TU/e. 2007-12

A.J. Wijs. *What to do Next?: Analysing and Optimising System Behaviour in Time*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13

C.F.J. Lange. *Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML*. Faculty of Mathematics and Computer Science, TU/e. 2007-14

T. van der Storm. *Component-based Configuration, Integration and Delivery*. Faculty of Natural Sciences, Mathematics, and Computer Science,UvA. 2007-15

B.S. Graaf. *Model-Driven Evolution of Software Architectures*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16

A.H.J.Mathijssen. *Logical Calculi for Reasoning with Binding*. Faculty of Mathematics and Computer Science, TU/e. 2007-17

D. Jarnikov. *QoS framework for Video Streaming in Home Networks*. Faculty of Mathematics and Computer Science, TU/e. 2007-18

M. A. Abam. *New Data Structures and Algorithms for Mobile Data*. Faculty of Mathematics and Computer Science, TU/e. 2007-19

W.Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy*. Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS*. Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems*. Faculty of Mechanical Engineering, TU/e. 2008-05