Model-based system analysis using Chi and UPPAAL : an industrial case study

Systems Engineering Group
Department of Mechanical Engineering
Eindhoven University of Technology
PO Box 513
5600 MB Eindhoven
The Netherlands
http://se.wtb.tue.nl/

SE Report: Nr. 2006-07

Model-based system analysis using χ and UPPAAL: an industrial case study

N.C.W.M. Braspenning, E.M. Bortnik,
J.M. van de Mortel-Fronczak, J.E. Rooda

ISSN: 1872-1567

SE Report: Nr. 2006-07
Eindhoven, November 2006
SE Reports are available via http://se.wtb.tue.nl/sereports

Abstract

New methods and techniques are needed to reduce the integration and test effort (lead time, costs, resources) in the development of high-tech multi-disciplinary systems. To facilitate this effort reduction, a method called model-based integration and testing is being developed. The method allows to integrate formal executable models of system components that are not yet physically realized with available realizations of other components. The combination of models and realizations is then used for early analysis of the integrated system by means of validation, verification, and testing. The analysis enables early detection and prevention of problems that would otherwise occur during real integration, resulting in a significant reduction of effort invested in the real integration and testing phases. This paper illustrates the application of the method to a realistic industrial case study, focusing on verification of the models obtained. We show how a system model has been developed for model-based integration and testing in the timed process algebra χ (Chi), and how certain behavioral properties of this model have been verified by the UPPAAL model checker.

1 Introduction

High-tech multi-disciplinary systems like wafer scanners, electronic microscopes and high-speed printers are becoming more complex every day. Growing system complexity also increases the effort (in terms of lead time, cost, resources) needed for the, so-called, *integration and testing phases*. During these phases, the system is integrated by combining component realizations and, subsequently, tested against the system requirements. Existing industrial practice shows that the main effort of system development is shifting from the design and implementation phases to the system integration and testing phases [1]. Furthermore, finding and fixing problems during integration and testing can be up to 100 times more expensive than finding and fixing the problems during the requirements and design phases [2].

Literature reports wealth of research proposing a *model-based* way of working to counter the increase of development effort, like requirements modeling [3], model-based design [4], model-based code generation [5], hardware-software co-simulation [6], and model-based testing [7]. In most cases, however, these model-based techniques are investigated in isolation, and little work is reported on combining these techniques into an overall method. Although model-based systems engineering [8] and OMG's model-driven architecture [9] (for software only systems) are such overall model-based methods, these methods are mainly focusing on the requirements, design, and implementation phases, rather than on the integration and test phases. Furthermore, literature barely mentions realistic industrial applications of such methods, at least not for high-tech multi-disciplinary systems.

Our research within the TANGRAM project [10] focusses on a method of *model-based integration and testing* (MBI&T), introduced in [11]. In this method, formal executable models of system components (e.g. software, mechanics, electronics) that are not yet realized are integrated with available realizations of other components, establishing a *model-based integrated system*. Such a model-based integrated system can be established much earlier compared to a real integrated system, and it can effectively be used for early model-based system analysis and system testing, which has three main advantages. First, the fact that it is earlier means that the integration and test effort is distributed over a wider time frame. This reduces the effort to be invested during the real integration and testing phases. Secondly, it allows earlier and thus cheaper detection and prevention of problems that would otherwise occur during real integration. Early problem detection and prevention also reduces the corresponding diagnostic and fix effort and increases the quality of the system at an earlier stage. Finally, the use of formal models enables the application of powerful model-based analysis techniques, like simulation and verification. These analysis techniques help to improve the insight in the system's behavior for the engineers, resulting in better system quality as well.

This paper illustrates the application of the MBI&T method to the development of a part of a realistic industrial system, namely the ASML [12] wafer scanner.

In the case study, a system is formally specified by means of a process algebraic language χ (Chi) [13]. The χ language is supported by a toolset that allows simulation of the obtained χ model, e.g. for the analysis of system performance and exceptional behavior handling. The formal semantics of χ enable functional analysis of χ models, e.g. verification of the correctness of a system model. Combining performance analysis and functional analysis in χ environment is the objective of the TIPSY project [14]. As a part of this project, a translation scheme [15] from χ to UPPAAL timed automata [16, 17] has been developed and integrated into the χ toolset, allowing verification of the translated models by UPPAAL model checker. The UPPAAL tool has been used in a number of industrial case studies. The complete overview can be found in [18]. Mostly, the case studies concerned verifying real-time controllers [19] and communication protocols [20, 21]. In [22] the problem of synthesizing production schedules and control programs for the mock-up of the batch production plant was addressed. In [23] the throughput of an ASML wafer scanner was analyzed with UPPAAL.

The case study described in this paper focuses on verification of system properties such as deadlock freeness, liveness, safety, and temporal properties using the χ to UPPAAL translation scheme and UPPAAL model checker. The goal of the case study and this paper is threefold. We show the potential of the proposed MBI&T method (in which the verification techniques

are used) to reduce the integration and test effort of industrial systems. We investigate the applicability, scalability, and usability of the χ toolset as integrated tool support for all aspects of the MBI&T method, particularly focusing on the implementation of the χ to UPPAAL translation scheme. Finally, we investigate the applicability and the advantages of using verification techniques for industrial systems.
The structure of the paper is as follows. Section 2 describes the MBI&T method in general with the focus on the use of verification within the method. Section 3 introduces the industrial case study to which the MBI&T method has been applied. The activities that have been performed in the case study are described in Section 4 (modeling and simulation with χ) and Section 5 (translation to and verification with UPPAAL). Although model-based and real system testing are not the focus of this paper, Section 6 gives a short summary of these steps. Finally, the conclusions are drawn and discussed in Section 7.

2 Model-based integration and testing method

In current industrial practice, the system development process is subdivided into multiple concurrent component development processes. Subsequently, the resulting components (e.g. mechanics, electronics, software) are integrated into the system.
The development process of a system S that consists of n components $C_{1..n}$ (we denote a set $\{A_1, \ldots, A_i, \ldots, A_n\}$ by $A_{1..n}$) starts with the system requirements R and system design D. After that each component is developed. The development process of a component C_i consists of three phases: requirements definition, design, and realization. Each of these phases results in a different representation form of the component, namely the requirements R_i, the design D_i, and the realization Z_i. The realization of system S is the result of the integration $\{Z_{1..n}\}_I$ of realizations $Z_{1..n}$ by means of the infrastructure I. Here, infrastructure I contains everything that is needed to connect the components in order to let them perform the system's function, e.g., nuts and bolts (mechanical infrastructure), cables (electronic infrastructure), communication network (software infrastructure).
Fig. 1 shows a graphical representation of the development process of system S. The development phases are depicted by arrows and different representation forms of systems and components are depicted by boxes. For simplicity, the figure shows a 'sequential' development process, however in practice the development process will have an incremental and iterative nature. This involves 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.

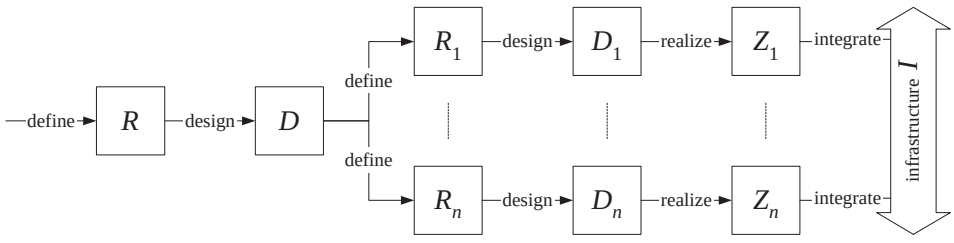

Figure 1: Current system development process

In this way of working, only two types of system level analysis can be applied. On the one hand, the consistency between requirements and designs on the component level and on the system level can be checked, i.e. $R_{1..n}$ versus R and $D_{1..n}$ versus D, which usually boils down to reviewing lots of documents. On the other hand, the integrated system realization $\{Z_{1..n}\}_I$ can be tested against the system requirements R, which requires that all components are

realized and integrated. This requirement means that if problems occur and need to be fixed during the integration and test phases, the effort invested in these phases increases and on-time shipment of the system is directly threatened. If integration problems could be detected and prevented at an earlier stage of development, the effort invested in the integration and test phases would be reduced and valuable test time would be saved. As a result, the system could be shipped earlier (which is critical in the lithography industry), or the saved test time could be used to further increase the system quality.

In [11], we introduced the MBI&T method to reduce the integration and test effort. This method takes the design documentation of the components C_i as a starting point and represents them by formal executable models M_i (Fig. 2). The requirements documentation is used to formulate the properties of the system and components. An infrastructure I is used that allows the integration of models $M_{1..n}$ and realizations $Z_{1..n}$, such that all possible combinations of models and realizations can be integrated. As an example, assume that $n = 2$, i.e. the depicted components C_1 and $C_n = C_2$ are the only components of the system. Then Fig. 2 shows, corresponding to the depicted integration 'switches', the model-based integrated system $\{M_1, Z_2\}_I$. In the MBI&T method, the infrastructure can be, for example, parallel composition (to connect component models) or specific software/hardware infrastructure (to connect models and realizations). Note that in this paper, we do not focus on the integration of models and realizations. Also, we do not discuss the corresponding specification and implementation issues of infrastructure I, which is part of our current work and will be reported in the future.

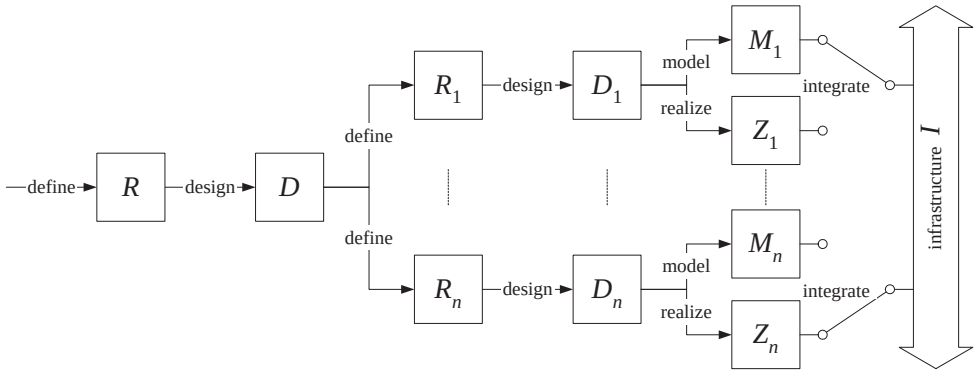

Figure 2: System development process in the MBI&T method

In principle, the MBI&T method allows the use of different specification languages and tools, as long as they are suitable for modeling, analysis, and testing of the considered aspects of the considered components. In the presented case study, all components of the system are modeled in the process algebraic language χ [24, 25]. The χ language is intended for modeling, simulation, verification, and real-time control of discrete-event, continuous or combined, so-called hybrid, systems, such as manufacturing systems. The toolset [26] allows modeling and simulation of χ models, as well as their translation to different formalisms. The χ language and simulator have been successfully applied to a large number of industrial cases, such as integrated circuit manufacturing plants, process industry plants, and machine control systems [27, 28]. Recently the χ toolset was extended with the translator to UPPAAL.

When all components of a system have been modeled as χ processes, the integrated system model $M_{1..n}$ can be obtained by parallel composition of the χ processes, $(M_1 \parallel \ldots \parallel M_n)$, where the parallel composition models the infrastructure I.

For the analysis of the integrated system model $\{M_{1..n}\}_I$, several model-based techniques can be applied. For instance, the χ simulator can be used to simulate specific behavior scenarios

of the system. The simulation results, then, can be compared with the intended system design D and requirements R.
Due to the complexity of the industrial systems, which often involve both high-level parallelism and non-deterministic behavior, simulation can only show that a system model *might* have correct behavior. To prove the correctness of a system model in general, verification can be used. One of the most popular verification techniques is model checking, which allows to prove automatically the validity of a given property (derived from the system requirements R and system design D), for a given model of a system.
The translator from χ to UPPAAL enables automatic translation of χ models to UPPAAL timed automata, which can subsequently be verified using the UPPAAL model checker. Model checking of χ models with UPPAAL is the main topic of this paper.
With simulation and verification, it can be determined whether the system model is a correct representation of the intended system design and whether it satisfies certain properties. If this is the case, and certain component realizations become available, the models of the components can be used for both model-based component testing and for model-based system testing.
Model-based component testing involves automatic testing of the realization of a component, Z_i, against a model of that component, M_i. Using techniques and tools from model-based testing research [7], test cases are automatically generated from the model and automatically executed on the realization. Model-based component testing using χ models and the model-based testing tool TORX [29] has been reported in previous work [11].
While model-based component testing focuses on single components, model-based system testing focuses on *integrated* models and realizations. Here, the available realizations and the models are coupled via an appropriate infrastructure I, using specific software and hardware tooling. The resulting model-based integrated system, e.g. $\{M_1, Z_2\}_I$ for $n = 2$, is tested on system level using test cases derived from the system requirements R and the system design D. Since this does not require that all component realizations are available, and since models are usually available earlier than realizations, model-based system testing enables earlier detection and prevention of problems when compared to real system testing. Furthermore, models allow easier testing of exceptional behavior, because the model behavior can easily be adapted to create exceptional conditions, for example, a broken component. These advantages of model-based system testing both contribute to a reduction of the effort invested during real integration and testing.
The case study described in the following section illustrates the application of the MBI&T method to an industrial system.

3 Case study: ASML EUV machine

To show proof of concept and to evaluate the MBI&T method, particularly focusing on the verification of χ models using UPPAAL, the method was applied to a part of an ASML wafer scanner [12]. In an ASML wafer scanner, laser light transfers a lithographic image onto the surface of a silicon wafer with nanometer accuracy. The laser light passes through an optical system that scales down the pattern image before it is projected onto the wafer. Currently, a new type of wafer scanner is under development within ASML, which uses extreme ultra violet (EUV) light for exposing wafers. One of the most important technical challenges in the development of this lithography system is the need for strict vacuum conditions, since EUV light is is absorbed by nearly all materials, including air.
In the case study presented in this paper, the focus is on the interaction between the vacuum system component C_v that controls the vacuum conditions and the source component C_s that generates the EUV light. These components need close cooperation 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.

Fig. 3 shows the components and interfaces involved in the case study. 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 latches[1], 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 right 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 the latch interface with the source, the vacuum system has another interface to communicate with the environment C_e, e.g. a control component or a vacuum system operator. The environment can request the vacuum system to go to either the vacuum or the vented state by sending a 'request' message. After handling a request, the vacuum system notifies the environment by sending a 'reply' message.

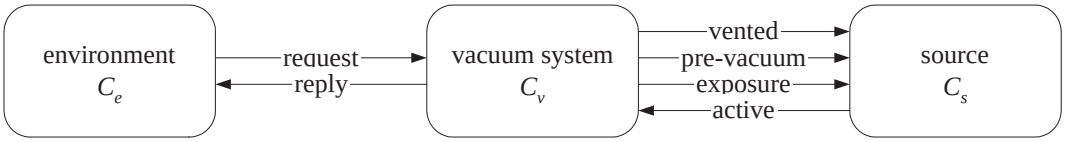

Figure 3: Components and interfaces involved in the case study

The behavior of the integrated system under nominal conditions is depicted in message sequence chart (Fig. 4). Initially, the vacuum system is vented (accordingly, the 'vented' latch is active) and the source is inactive. When the vacuum system receives a request from the environment to go to the vacuum state it deactivates the 'vented' latch and starts the vacuum pumps. The source observes that the 'vented' latch is inactive and executes some initial preparation steps. While pumping down, the vacuum system measures the vacuum conditions. When the vacuum conditions are sufficient to activate the source (the pre-vacuum conditions are reached), the vacuum system activates the 'pre-vacuum' latch. When the source observes this signal, it first activates the 'active' latch, and subsequently performs the necessary actions to reach the active state. After further pumping down, when the vacuum conditions are right for exposure, the vacuum system activates the 'exposure' latch. Then, via the 'reply' interface the vacuum system notifies the environment that it is in the vacuum state. After the source observes the active 'exposure' latch, it goes to the exposure state and both components are ready for exposure. For the other way around, going from vacuum/exposure conditions to vented/inactive conditions, a similar, reversed sequence is specified.
The nominal sequence described above does not cover preemption. The environment can interrupt the sequence at any time by a new request, and the vacuum system should handle these interrupts. For instance, 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 described above), the vacuum system should immediately interrupt the vacuum sequence and start with the venting sequence to go to the vented state.
Finally, errors can be raised by the source if, for instance, an unexpected communication event occurs. The severity of an error is indicated by error levels. Depending on the severity

[1] Latch: electronic circuit with inputs 'set' and 'reset' that is capable of storing one bit of information, i.e., a high or a low voltage

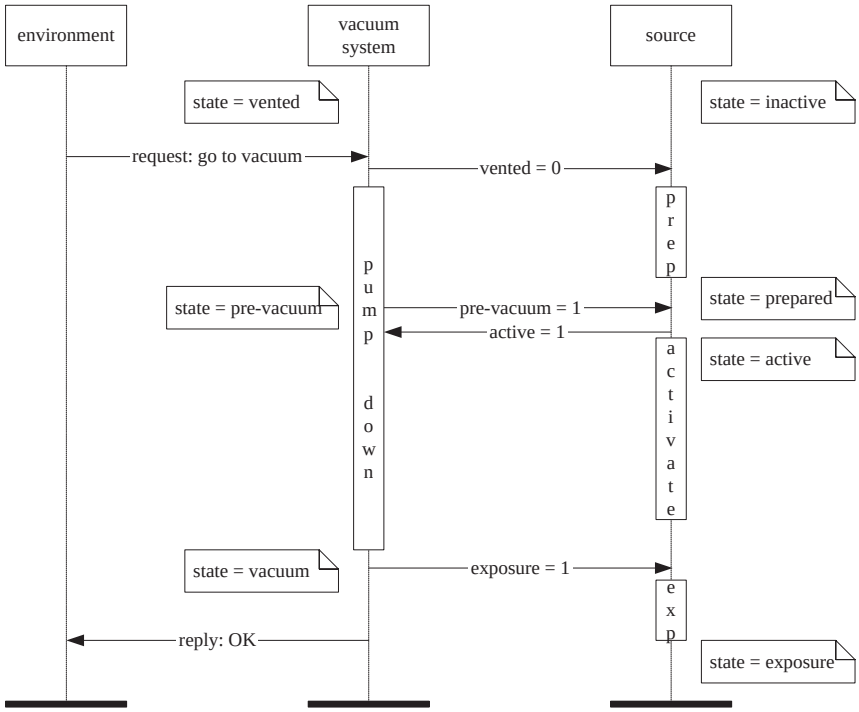

Figure 4: Nominal system behavior for the vacuum sequence

of an error, the source might need to perform certain actions to avoid further problems. For example, the source should leave the expose state but remain active for a low level error, however it should go to the inactive state as soon as possible in the case of a high level error. The most important requirements to the system are:

1. The actions and communications in the vacuum and venting sequences are executed in correct order.
2. The behavior of the system can be interrupted at any time, and these interrupts are handled correctly.
3. The source does not raise unnecessary errors, i.e. not during nominal behavior.
4. The source may not be active while the vacuum system is vented in order to prevent machine damage.
5. The duration of the vacuum and venting sequences is at most 6 hours and 1 hour, respectively.

Using the obtained design documentation of the components as a starting point, the MBI&T method was applied according to the following steps, corresponding to Fig. 5:

Step 1: Modeling of the system components as χ processes M_e, M_v, and M_s, based on the ASML design documents D_e, D_v, and D_s.

Step 2: Model-based analysis of the integrated system model $\{M_e, M_v, M_s\}_I$, modeled as $(M_e \parallel M_v \parallel M_s)$:

a. Simulation using the χ simulator, i.e. analyzing certain scenarios derived from R and D of $\{M_e, M_v, M_s\}_I$ and comparing the resulting behavior against R and D.

b. Verification using the UPPAAL model checker, i.e. translating the χ system model to UPPAAL timed automata and verifying all traces of $\{M_e, M_v, M_s\}_I$ against properties derived from R and D.

Step 3: As soon as the realization of a component becomes available (in the case study, this was the source realization Z_s):

a. Model-based component testing of the component's realization with respect to its model, i.e. Z_s with respect to M_s, using automatic model-based testing techniques and tools, e.g. TORX.

b. Replacing the model of the source M_s by the source realization Z_s using an infrastructure I that enables the integration of Z_s with the models M_e and M_v. This results in the model-based integrated system $\{M_e, M_v, Z_s\}_I$.

c. Model-based system testing of $\{M_e, M_v, Z_s\}_I$ using test cases derived from R and D.

Step 4: After all models have been substituted by realizations: testing of the complete system realization $\{Z_e, Z_v, Z_s\}_I$ by executing test cases derived from R and D. Note that only this step is performed in the current system development process as well.

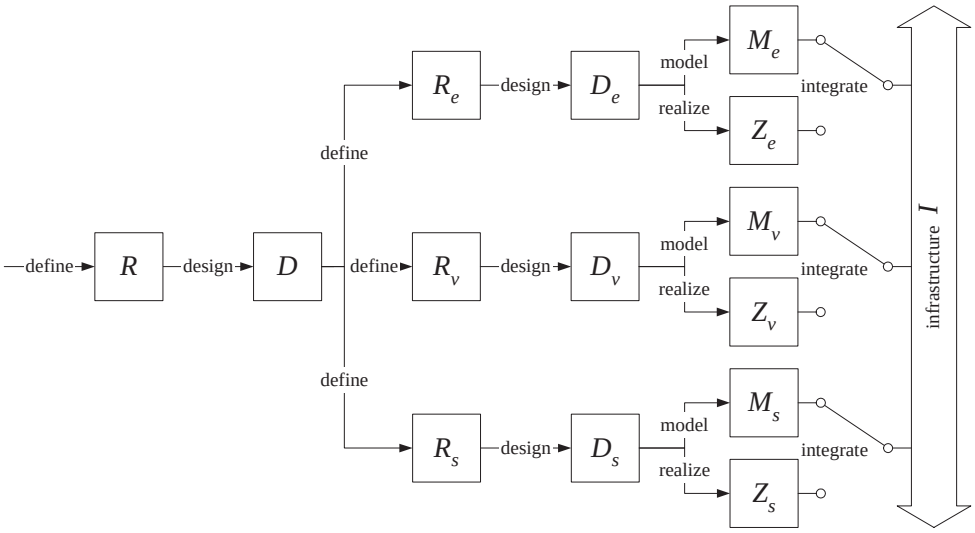

Figure 5: MBI&T method applied to the case study

In this paper, we particularly focus on the application of steps 1, 2a, and 2b. Furthermore, we give a summary of the application of steps 3b and 3c and the effects of applying the MBI&T method on the real integration and testing in step 4. Application of step 3a was not performed in the case study, although a similar case study involving model-based component testing has been performed and reported in [11].

4 Modeling and simulation with χ

As mentioned in Section 2, a system development process will be more incremental and iterative in nature than it is depicted in Fig. 5. This was also experienced in the case study, especially during the modeling (step 1) and simulation (step 2a) activities described in this section.

Step 1: Modeling the components in χ

The informal design documentation was taken as a starting point for modeling the vacuum system and the source as χ processes. The system level design documentation (corresponding to D) described the interfaces between the vacuum system and source as shown in Fig. 3. The design documentation for the source component, D_s, was rather complete and contained a good overview of the behavior in the form of a state diagram including most of the possible actions and communications. However, the design documentation of the vacuum system component, D_v, only described the internal actions for the vacuum and venting sequences, and did not contain information about the communication with the source. 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 described the nominal behavior only 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 resulting design specifications were validated for their correctness by discussion with the designers of the vacuum system and source.

We experienced that in most cases, the issues that arose during the modeling activities in fact indicated unknown or incomplete design issues, like missing states, obsolete states, and errors in the communication sequence. By incremental modeling, intermediate simulation, discussion, and design review, the system specification was further corrected and completed, which also helped the engineers to obtain a better system overview.

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 be experienced in the real system. Therefore, the behavior concerning these issues has been abstracted in the model by putting an explicit indication of 'undefined behavior'. In step 2b of the case study, it will be verified whether this undefined behavior indeed can never be experienced.

The next paragraphs describe how the source C_s, the vacuum system C_v, and the environment C_e were modeled as χ processes. Fig. 6 shows the process layout of the system model, which is based on the system design layout from Fig. 3. In the figure, the environment is modeled as a single sequential process M_e, the vacuum system M_v is modeled as a parallel composition of the processes ($v1 \parallel v2 \parallel v3$), and the source M_s is modeled as the parallel composition of the processes ($s1 \parallel s2 \parallel s3 \parallel s4$). The arrows depict the channels that model the communication between processes of different components, and the bold lines depict variables that are shared between processes of one component.

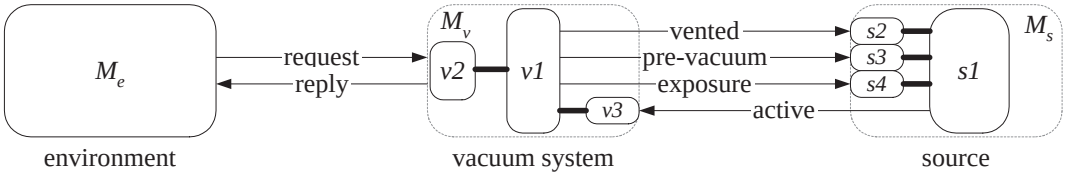

Figure 6: Process layout of the χ system model

As an example of a χ process, Fig. 7 shows a part of the source model M_s. For simplicity, the declaration and initialization of the variables are omitted and only a part of the core process $s1$ is shown. The processes $s2$, $s3$, and $s4$ model the interaction with the vacuum system via

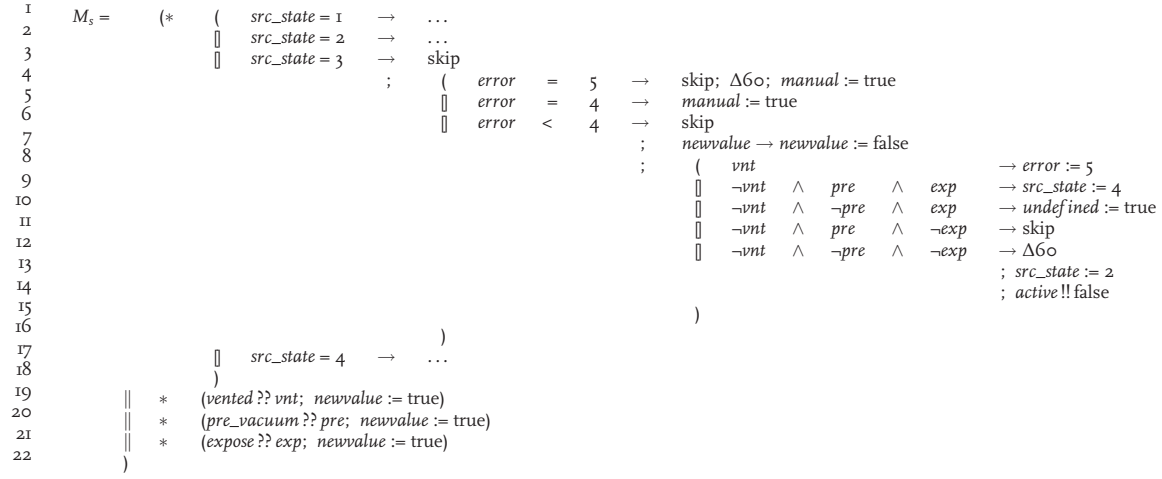

Figure 7: Part of the source model M_s in χ

the latches 'vented', 'pre-vacuum', and 'exposure', respectively. In Fig. 7, $s1$ is shown on lines 1-18, and $s2$, $s3$, $s4$ are shown on lines 19, 20, 21, respectively.
The core process $s1$ specifies the internal behavior of the source. The state of process $s1$ is modeled by the variable *src_state*, which can take on the value 1 (inactive), 2 (prepared), 3 (active) or 4 (exposure). 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. The boolean variables *vnt, pre, exp* indicate the states of the corresponding latches. When the vacuum system sends a signal via a latch channel, the new value is assigned to the corresponding variable *vnt, pre* or *exp* in the processes $s2$, $s3$, and $s4$, respectively. The variable *newvalue* indicates that the state of some latch was changed. Finally, the variables *manual* and *undef ined* indicate that the source has reached a state where it is in manual mode or for which the behavior is undefined.

The core process repetitively checks its state and performs the corresponding actions. As an example let us consider the case if the source is in its active state, which is the part of $s1$ shown in Fig. 7. In this case the guard *src_state* = 3 is true and the process makes an undelayable internal action (skip). After that the process checks if there is an error and, depending on the severity of the error, it takes certain actions to prevent further problems. If *error* = 5, the source performs a delay for 60 time units (Δ60) and then switches to manual mode (*manual* := true). The delay of 60 time units models the time which is needed to process the error. If *error* = 4, the source immediately switches to manual mode.
If there is no severe error (*error* < 4), the source process performs an internal action (skip). Subsequently, it checks if any of the latch values has been changed, indicated by the variable *newvalue*. If *newvalue* is true, it is reset to false without a delay and the process continues. 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).
Subsequently, the current states of all four latches are checked. 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 4, the exposure state. If *pre* is false and *exp* is true, the behavior is undefined, *undef ined* := true. If *pre* is true and *exp* is false, the process performs an internal action skip and remains in the active state. If both *pre* and *exp* are false, the process delays for 60 time units, modeling the tasks needed to go back to the prepared state (*src_state* := 2). After that the source process $s1$ deactivates the 'active' latch channel to $v3$ of the vacuum system. This is modeled by sending the value false via the channel *active* (*active* !! false).

The vacuum system component C_v is modeled as a χ process M_v, and consists of three parallel processes (Fig. 6). The process $v1$ is the core process and describes the internal behavior of the vacuum system, similar to process $s1$ of M_s.
The process $v2$ models the interaction with the environment. Receiving requests from the environment is modeled by communication actions along the channel *request*. Upon receiving a new request, $v2$ decides which actions should be performed, e.g. start a new sequence or interrupt the current sequence. Accordingly, $v2$ changes the corresponding variables shared with $v1$.
The process $v3$ models the interaction with the source via the 'active' latch. It receives signals from the source via the channel *active* and changes its state correspondingly. Similar to the source model M_s, the state of the latch is modeled by the variable shared with $v1$.
The environment component C_e is modeled as a χ process M_e. It can send a request to change the state of the vacuum system via the channel *request*, and receive a reply via the channel *reply* (Fig. 6). For the analysis in the next steps of the case study we model the environment M_e as a generic environment 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 will be used, while for verification, all scenarios (with any possible delay) will be analyzed.
Finally, the system model $\{M_e, M_v, M_s\}_I$ is modeled in χ as the parallel composition of the component processes: $(M_e \parallel M_v \parallel M_s)$. The parallel composition synchronizes the components on time and on communication.

Step 2a: Simulation of the integrated system model

In this step, the goal is to inspect the behavior of the integrated vacuum system-source model by means of simulation.
The simulation experiments require different scenarios to analyze different aspects of the system. A good source for possible scenarios is the intended system behavior specified in the system requirements and design documentation, R and D. Unfortunately, in the case study only one scenario could directly be derived from the documentation. 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 most cases, it is the exceptional behavior which gives the problems, since this behavior is less documented and thus less clear when compared to the nominal behavior. Therefore, it is very important to analyze the exceptional behavior as soon as possible.
In this case study, the exceptional behavior of the system is also poorly documented. No simulation scenarios could be directly derived from the documentation. However, based on our system overview obtained by modeling the components, and by discussion with the involved engineers, four additional scenarios for exceptional behavior analysis have been 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 way of working. Since only realizations can be used for system analysis, it may be difficult or expensive to create the non-nominal circumstances that are necessary to analyze the exceptional behavior, for example, a broken component. Since the MBI&T method uses models for system analysis, creating these non-nominal circumstances is much easier and cheaper.
In the case study, we used specific configurations of the 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.
The simulation results revealed one situation with incorrect behavior. This situation 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, since the vacuum system has been designed to deactivate both latches at the same time, 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.

5 Translation to and verification with UPPAAL

During simulation (validation) some problems were discovered and solved. That increases the confidence, but does not prove the correctness of the model. To check whether the model behaves correctly in all possible scenarios and to gain more knowledge about the system, it has to be verified. This corresponds to step 2b of the MBI&T method.

Step 2b: Verification of the χ system model using UPPAAL

UPPAAL is a tool for modeling, simulation, and verification. A system, modeled as a network of UPPAAL timed automata, can be simulated by UPPAAL simulator and verified by UPPAAL model checker. To be able to verify χ models in UPPAAL, the translation scheme from the process algebraic language χ to UPPAAL timed automata has been formally defined and the proofs of its correctness have been given [30]. The translation scheme is defined for a subset of χ and consists of all χ models, specified as parallel composition of one ore more sequential processes. The translatable subset of χ 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, in χ both undelayable ($h\,!!\,\mathbf{e}_n$) and delayable ($h\,!\,\mathbf{e}_n$) send process terms are defined. Delayable send is a syntactic extension, which is 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. While performing this case study, we also wanted to check if the defined subset is expressive enough.
- Make the translation as general as possible. For instance, we could not define a general translation of the guard operator, but there are many specific cases, for which guarded process terms can be translated. We limit ourselves to guarded skip, multi-assignment, send and receive.
- Make the translated UPPAAL models as readable as possible. For example, nested parallel composition can be translated as alternative composition of all possible transitions [24], but it would make resulting automata less readable.

The translation scheme from χ to UPPAAL has been implemented and integrated into the χ toolset, enabling automatic translation of χ models to UPPAAL timed automata [26].

Making the χ model translatable

Since the original model was created without considering its translation to UPPAAL, it uses some constructs, for which translation is not defined, such as modeling scope operator, process instantiation, delayable send and receive, nested parallelism, and guarded delay. That means that the model has to be transformed to make it translatable.
The modeling scope operator is a syntactic extension, which is used to declare a scope, consisting of local variable, channels, etc. Although in UPPAAL there are global and local scopes, they are not defined formally, and so the modeling scope operator cannot be translated. The same holds for the process instantiation. To remove these operators from the χ model, all the variables have been lifted to the global scope and given unique names. After that the local scope operators and the process instantiations can be safely removed.

All delayable send and receive process terms ($h\,!\,\mathbf{e}_n$, $h\,?\,\mathbf{x}_n$) have been replaced by their definitions ($[h\,!!\,\mathbf{e}_n]$, $[h\,??\,\mathbf{x}_n]$). The behavior of the model is not changed.
In the original model guarded delays have been used in following construct: $b \rightarrow a \;[]\; \neg b \rightarrow \Delta d$, where $a \in \{\text{skip}, \mathbf{x}_n := \mathbf{e}_n\}$. It can be shown that this process term is bisimilar to $b \rightarrow a \;[]\; \Delta d$, in a way, similar to the proofs of the χ properties in [24].
The vacuum system process contains a construct of the form $p_1; (p_2; p_3 \parallel p_4); p_5$. Since nested parallelism is not allowed, this part of the model has to be re-written. Here, the process p_2 changes a value of a variable x. As soon as it happens, the process p_4 should perform an undelayable action and terminate. The analysis of the system has shown that this particular case can be also modeled without usage of the nested parallel operator as $p_1; p_2; p_4; p_3; p_5$.
The case study has shown that the translatable subset of χ should be extended with the translation of modeling scope operator and process instantiation. Since it is not possible to make a general translation of nested parallelism, guarded delay and real-valued delays, these constructs have to be avoided while creating a model for verification, otherwise they have to transformed manually.
After transforming the original χ model, the corresponding translatable model has been translated automatically to UPPAAL automata.
Fig. 8 shows the generated UPPAAL automata for the source, which corresponds to the partial χ source model M_s of Fig. 7.

Verification of the generated UPPAAL model

The following properties of the resulting UPPAAL model have been verified:

(1) Deadlock freeness: `A[] deadlock imply env.end`, where `env.end` denotes the location of the environment automaton that indicates successful termination.

(2) Livelock freeness: `A<> env.end`. The system model is created in such way that the vacuum system and the source components get requests from the environment component; when all requests are processed and confirmation is received, the environment process terminates. Based on this we can state that if the environment automaton reaches its end state, there is no livelock in the system.

(3) No undefined behavior: `A[] undefined == 0`. While modeling we discovered that in some particular situations the system behavior was unknown, since it can never occur. These situations were modeled by assigning a non-zero value to the variable `undefined` (Fig. 7, line 10).

(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) Vacuum system may not be vented while 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 vacuum 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 `vented` 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 has been verified in UPPAAL using the following options: generation of the fastest trace, breadth first search order, conservative space optimization, and state space representation uses minimal constraint systems. The biggest number of states (20510) was explored while verifying the first property.
During verification of properties 1 and 2, two design errors have been found. Both errors are causing deadlock and concern non-determinism in the interleaving of the main process and the interrupt handling process of the vacuum system (v1 and v2 in Fig. 6, respectively). The way to handle this non-determinism in general has not been specified in the design

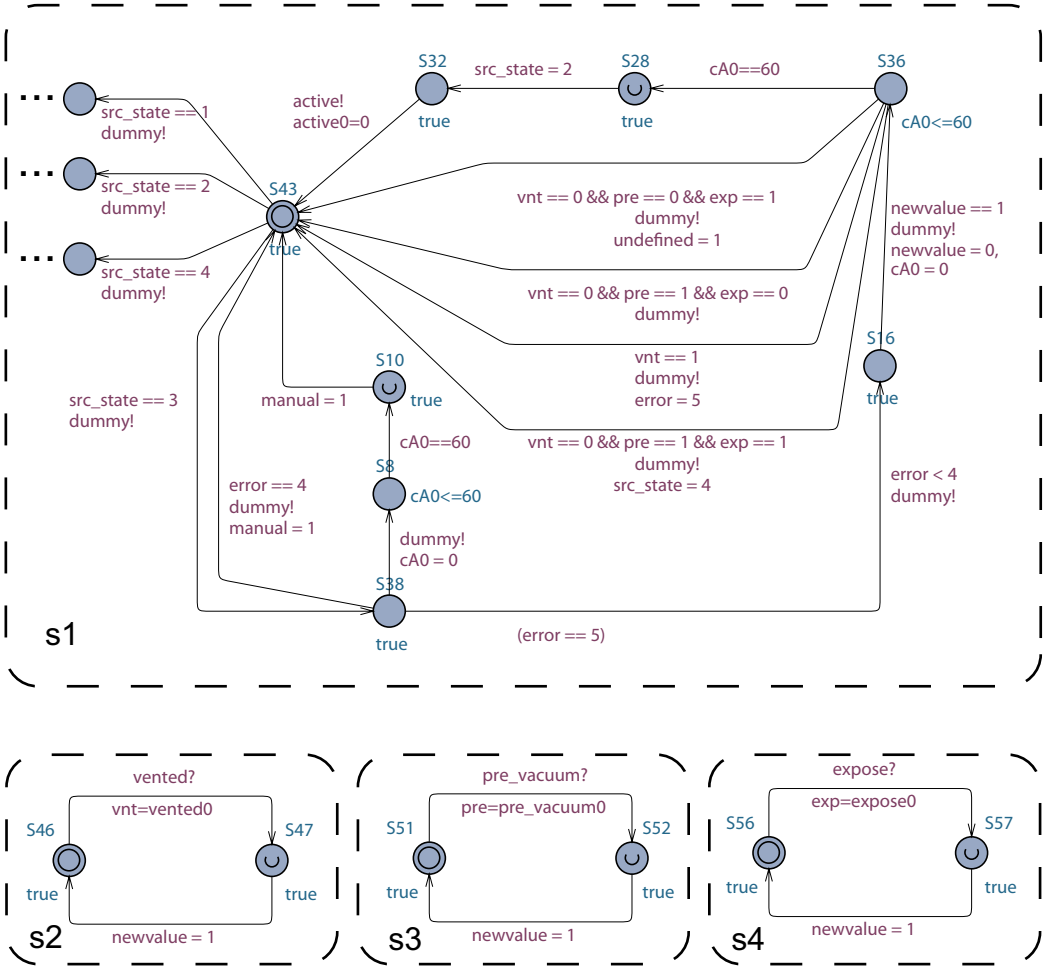

Figure 8: Generated UPPAAL automata for the source

documentation, and model verification has indicated this design incompleteness. After informing the involved engineers, an alternative design for the interrupt handling process has been proposed.

This alternative design has also been applied to the χ model (and subsequently translated to UPPAAL). Besides this general issue, another deadlock occurred in the specific situation where the 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. With the alternative design for the interrupt handling process, and after restricting the test environment such that interrupts can only occur after the start of a sequence, the model satisfies both properties 1 and 2.

Property 3 is satisfied by the model, indicating that the engineers were correct when they claimed that the undefined behavior parts would never be reached. Verification of property 4 detected the same design error as found by simulation (incorrect behavior in the venting sequence, resulting in an error level larger than zero). Finally, two minor errors have been 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 the property 6, a method similar to the decoration method described in [19] was used. We added the boolean variables **vacuum** and **vented** to indicate which sequence is

being performed and a clock clk to determine the duration of a sequence. Whenever the environment sends a request to the vacuum system to start a sequence, the corresponding variable is set to true and the clk is reset. When the sequence is finished, the vacuum system sends a reply back to the environment and both variables are set to false. No new edges, locations, invariants or guards were added. It can be shown that the behavior of the system was not changed. The properties A[] vacuum imply clk ≤ 21600 and A[] venting imply clk ≤ 3600 are both satisfied.
All design errors found with simulation and verification have been discussed with the ASML engineers, and subsequently fixes have been applied to the design and, correspondingly, to the model in order to avoid these errors. The fixed model has been verified again and now all properties as described above are satisfied by the model. For the fixed model the biggest number of states (9961) was explored while verifying the first property. The difference in state space size between the original model (20510 states) and the fixed model (9961 states) is partially caused by the fixes applied to the model, and partially by making use of an improved version of UPPAAL. UPPAAL 3.6 beta was used for the original model, and UPPAAL 4.0.2 was used for the fixed model.
The verification results of the fixed model give enough confidence that the model is a correct representation of the system design, making it suitable for model-based system testing.

6 Model-based and real system testing (summary)

Although model-based system testing and real system testing are not the focus of this paper, this section gives a summary of these steps of the case study.
The model of the vacuum system was integrated with the realization of the source using an appropriate infrastructure, including a software/hardware adapter that translates communication actions in the models into electronic signals for the source realization and vice versa. For model-based system testing, the same environment model and scenarios as in the simulation step were used, however now the scenarios and models were executed in real-time (using a real-time χ simulator), since the source realization, by definition, is also running in real-time. With this setup, early system testing was performed 20 weeks before real integration, in a less stressful and much cheaper period of the development process.
Similar to the simulation step of the case study, creating non-nominal circumstances was easy in a model environment compared to a realization environment. In addition, setting up and switching between different test cases can be performed almost instantly when using models. Real system testing experience shows that this can take up to more than half of the total test time due to, for example, long setup times of the system and initialization errors in parts of the system that are not involved in the tests (which would be abstracted away in the system model). As a result, the tests performed took half a day of test time compared to an estimated four days of testing during the real integration and test phases.
Several integration problems in the source realization were discovered. The models improved the diagnosis of the errors, which appeared to be caused by implementation errors in the source. These errors would have caused several days of very expensive downtime in the clean room when they would be discovered during real system testing. Finally, no additional errors in the source realization were found (and no fixing was necessary) during real system testing, at least not for the aspects that were analyzed and tested using the MBI&T method.
These results clearly show that early model-based system testing using χ models that have been verified with UPPAAL has saved significant integration and testing time and costs in the case study.

7 Conclusions

The goal of the case study and this paper was threefold: (1) to show the potential of the proposed MBI&T method (in which the verification techniques are used) to reduce the in-

tegration and test effort of industrial systems; (2) to investigate the applicability, scalability, and usability of the χ toolset as integrated tool support for all aspects of the MBI&T method, particularly focusing on the translator from χ to UPPAAL; (3) to show the applicability and the advantages of using verification techniques for real-size industrial systems.

Application of the MBI&T method (and verification as its part) in the case study has shown many advantages. First, the modeling activities helped to clarify, correct, and complete the design documentation. By simulation and verification, a number of design and integration errors were detected and fixed earlier and cheaper when compared to current system development. Finally, a part of the model was integrated with a realized component to enable early, fast, and cheap model-based system testing. Again, multiple errors in realization were detected and fixed. This clearly shows the applicability and value of the MBI&T method for industrial system development.

In total, five errors have been found by simulation and verification of the system model. Three of these errors were design errors; only one of them has been discovered by means of simulation. The other two design errors, discovered by verification only, both concern the non-deterministic behavior of parallel processes. This is difficult, if not impossible, to understand and to analyze by simulation or reviewing the design documentation. This illustrates that verification should be used for designing real industrial systems, which often involve both high-level parallelism and non-deterministic behavior.

The other two errors were modeling errors introduced in the modeling and model transformation step. To prevent model transformation errors, the translation scheme from χ to UPPAAL should be extended with the scope operator and process instantiation. 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 constructs should be avoided while modeling; otherwise they have to be transformed manually.

The χ toolset, extended with the UPPAAL translator, is suitable for all activities of the MBI&T method: modeling, simulation, verification, component testing, and integrated system testing. The system aspects that were modeled in the case study (supervisory machine control in software, interrupt behavior, electronic communication) can easily be modeled in χ.

Although the state space explosion remains being the main obstacle in verification of the real-size industrial systems, the performed case study shows that continued research work and tools improvements allow to use model checking in wider application area. The fact that the state space of our system is rather small indicates that the χ toolset and the UPPAAL model checker are well suited for modeling and analysis of similar size or even more complex industrial systems. Our current work on the MBI&T method concentrates on the integration and testing of models and realizations, in particular on the behavior relation of the implemented infrastructure I and the parallel operator in the system model).

Acknowledgements

This work 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, and as part of the TIPSY project, supported by the Dutch Organization for Scientific Research (NWO), project number 612.064.205. The authors would like to thank the engineers of the ASML EUV group for their support in the case study, Ton Geubbels for his modeling and verification activities in the case study, and Esmée Bertens for her effort in transforming the χ model to make it suitable for translation to UPPAAL. Furthermore, we would like to thank all TANGRAM and TIPSY project members for their valuable comments and fruitful discussions.

Bibliography

[1] L. Bratthall, P. Runeson, K. Ädelsward, W. Eriksson, A survey of lead-time challenges in the development and evolution of distributed real-time systems, Information and Software Technology 42 (13) (2000) 947–958.

[2] B. Boehm, V. Basili, Software defect reduction top 10 list, IEEE Computer 34 (1) (2001) 135–137.

[3] M. Broy, O. Slotosch, From requirements to validated embedded systems, in: Embedded Software: First International Workshop, EMSOFT 2001, Tahoe City, CA, USA, Springer-Verlag, 2001, pp. 51–65.

[4] X. Liu, J. Liu, J. Eker, E. A. Lee, Heterogeneous modeling and design of control systems, in: Software-Enabled Control: Information Technology for Dynamical Systems, Wiley-IEEE Press, 2003, Ch. 7, pp. 105–122.

[5] F. J. Budinsky, M. A. Finnie, J. M. Vlissides, P. S. Yu, Automatic code generation from design patterns, IBM Systems Journal 35 (2) (1996) 151–171.

[6] J. A. Rowson, Hardware/software co-simulation, in: DAC '94: Proceedings of the 31st annual conference on Design automation, ACM Press, 1994, pp. 439–440.

[7] E. Brinksma, J. Tretmans, Testing transition systems: An annotated bibliography, in: MOVEP 2000 – Modelling and Verification of Parallel Processes, Vol. 2067 of Lecture Notes in Computer Science, Springer-Verlag, 2001, pp. 187–195.

[8] I. Ogren, On principles for model-based systems engineering, Systems Engineering 3 (1) (2000) 38–49.

[9] A. Kleppe, W. Bast, J. Warmer, MDA Explained: The Model Driven Architecture: Practice and Promise, 1st Edition, Addison-Wesley Professional, 2003.

[10] TANGRAM Project, Website, http://www.esi.nl/tangram.

[11] N. C. W. M. Braspenning, J. M. van de Mortel-Fronczak, J. E. Rooda, A Model-based Integration and Testing Method to Reduce System Development Effort, in: Proceedings of the 2nd workshop on Model-Based Testing (MBT2006), Electronic Notes in Theoretical Computer Science 164 (4) (2006) 13–28.

[12] ASML, Website, http://www.asml.com.

[13] D. A. van Beek, K. L. Man, M. A. Reniers, J. E. Rooda, R. R. H. Schiffelers, Syntax and consistent equation semantics of hybrid Chi, Journal of Logic and Algebraic Programming 68 (1-2) (2006) 129–210.

[14] TIPSy Project, Website, http://homepages.cwi.nl/~wijs/TIPSy.

[15] E. M. Bortnik, D. A. van Beek, J. M. van de Mortel-Fronczak, J. E. Rooda, Verification of timed Chi models using UPPAAL, in: ICINCO 2005, Proceedings of the Second International Conference on Informatics in Control, Automation and Robotics, Barcelona, Spain, September 14-17, INSTICC Press, 2005, pp. 486–492.

[16] G. Behrmann, A. David, K. G. Larsen, A tutorial on UPPAAL, in: Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004, no. 3185 in LNCS, Springer-Verlag, 2004, pp. 200–236.

[17] J. Bengtsson, W. Yi, Timed automata: Semantics, algorithms and tools, in: Lecture Notes on Concurrency and Petri Nets, no. 3098 in LNCS, Springer-Verlag, 2004.

[18] UPPAAL, Website, http://www.uppaal.com.

[19] M. Lindahl, P. Pettersson, W. Yi, Formal Design and Analysis of a Gearbox Controller, Springer International Journal of Software Tools for Technology Transfer (STTT) 3 (3) (2001) 353–368.

[20] J. Bengtsson, W. O. D. Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, W. Yi, Automated analysis of an audio control protocol using UPPAAL, Journal of Logic and Algebraic Programming 52–53 (2002) 163–181.

[21] A. David, W. Yi, Modelling and analysis of a commercial field bus protocol, in: Proceedings of the 12th Euromicro Conference on Real Time Systems, IEEE Computer Society, 2000, pp. 165–172.

[22] T. Hune, K. G. Larsen, P. Pettersson, Guided Synthesis of Control Programs Using UPPAAL, in: Proceedings of the IEEE ICDCS International Workshop on Distributed Systems Verification and Validation, IEEE Computer Society Press, 2000, pp. E15–E22.

[23] M. Hendriks, B. van den Nieuwelaar, F. Vaandrager, Model checker aided design of a controller for a wafer scanner, International Journal on Software Tools for Technology Transfer (STTT) - Special Section on Quantitative Analysis of Real-time Embedded Systems (2006) 1–15.

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

[25] D. A. van Beek, K. L. Man, M. A. Reniers, J. E. Rooda, R. R. H. Schiffelers, Syntax and semantics of timed Chi, Tech. Rep. 05-09, Eindhoven University of Technology, Department of Computer Science (2005).

[26] Systems Engineering Group, Eindhoven University of Technology, Chi language and tools website, http://se.wtb.tue.nl/sewiki/chi/start.

[27] D. A. van Beek, A. van der Ham, J. E. Rooda, Modelling and control of process industry batch production systems, in: 15th Triennial World Congress of the International Federation of Automatic Control, Barcelona, Spain, CD-ROM, 2002.

[28] J. M. van de Mortel-Fronczak, J. Vervoort, J. E. Rooda, Simulation-based design of machine control systems, in: Proceedings of the 15th European Simulation Multiconference, June 6-9, Prague, Czech Republic, 2001.

[29] J. Tretmans, E. Brinksma, TorX : Automated model based testing, in: 1st European Conference on Model-Driven Software Engineering, 2003.

[30] E. M. Bortnik, J. M. van de Mortel-Fronczak, J. E. Rooda, Verifying Chi models in UPPAAL, to appear in Systems Engineering Report (2006).