1

1a. Project Title: Modeling Control Aspects of Embedded Systems

1b. Acronym: MOCA

1c. Principal Investigator: Prof. Dr. R.J. Wieringa

2

2.1 Summary

Current verification technology has reached the maturity where it can be applied to non-trivial embedded software applications. However, producing a correctness proof for a system (the embedded software and its physical environnment) requires building a model of the system and this is still an art. The software part of a system can, in principle, be modeled automatically, but the physical part of the system must be modeled manually and informally. If we then formally prove a property of the model, we need a separate and essentally informal argument that the system has this property too. Our trust in the correctness of the system then partly depends on the quality of this essentially informal argument that the model accurately represents the system. Our aim in this project is to find techniques to enhance the quality of the model and of the informal argument that it accurately represents the system.

Our approach is to use joint decomposition of the system model and the correctness property, guided by the structure of the physical environment. The guidance provided by the physical environment is given by engineering blueprints that always exist of this environment, such as a P/I diagram. This decomposition is non-monotonic, because the proof that the lower-level properties jointly imply the composite correctness property may require the addition of axioms that force us to change the proofs already given. We will facilitate this process by identifying recurring patterns in the physical environment that can be reused in correctness proofs. The main deliverable will be a library of specification patterns with guidelines for reuse and tool support.

2.2 Abstract for Laymen (Dutch)

De huidige technologie voor formele verificatie van software heeft het stadium bereikt waarin het gebruikt kan worden voor de verificatie van de correctheid van embedded software. Het produceren van een correctheidsbewijs van embedded software is echter nog een slecht begrepen proces, waarin een model van de software en zijn fysieke omgeving gemaakt moet worden. Software is een formeel object dat in principle automatisch gemodelleerd kan worden, maar de fysieke omgeving is een informeel object waarvan het modelleren handmatig en informeel plaats vind. Als we dan bewijzen dat de door het model gerepresenteerde werkelijkheid een bepaalde eigenschap heeft, moeten we nog steeds aantonen dat het model accuraat is, d.w.z. dat het inderdaad de embedded software in zijn fysieke onmgeving representeert. Doel van dit project is het modelleringsproces en het accuraatheidsargument te verbeteren. Dit zal de bruikbaarheid van formele verificatietechnieken in de praktijk verbeteren.

Ons voorstel is om het modelleringsproces te verbeteren door de correctheidseigenschap en het omgevingsmodel samen te decomponeren, waarbij de decompositie een struktuur in de fysieke omgeving volgt, zoals die in de technische documentatie van die omgeving vastgelegd is. Na elke decompositie moet er bewezen worden dat de gevonden deeleigenschappen gezamelijk de oorspronkelijke correctheidseigenschap impliceren. Dit decompositieproces is

niet-monotoon, omdat na elke decompositie axioma's toegevoegd kunnen worden die er voor kunnen zorgen dat eerder gegeven bewijzen aangepast moeten worden. We zullen naar regelmatigheden in de fysieke omgeving zoeken die tot regelmatigheden in de gevraagde correctheidsbewijzen leiden, zodat we een bibliotheek van veel voorkomende decompositiestructuren en bijbehorende correctheidsbewijzen voor embedded software kunnen opbouwen.

3 Classification

Informaticasubdisciplines Verkenningscommissie 1996:

3. Software engineering

   3.2 Specificatiemethoden (requirements engineering)

   3.4 Testmethoden (formele verificatie)

NOAG-i:

- Embedded Systems
- Software Engineering

4 Composition of the Research Team

title	name	hours/week
Information Systems group (IS)		
Prof. Dr.	Roel Wieringa (Promotor)	4
Drs./Ir.	Ph.D. student 1	40
Distributed and Embedded Systems group (DIES)		
Dr.	Angelika Mader	4
Prof. Dr.	Pieter Hartel (Promotor)	2
Drs./Ir.	Ph.D. student 2	40
Formal Methods and Tools (FMT)		
Prof. Dr.	Ed Brinksma (Promotor)	2

5 Research Schools

The IS group is member of SIKS and the DIES and FMT groups are members of IPA.

6 Description of Proposed Research

6.1 Research goal

With the rapid spread of embedded software in mass-produced products such as consumer electronics and cars, and with the advent of mobile and ubiquitous technology, correctness of embedded software has become a prime concern of the manufacturers of these products. Current verification technology has reached the maturity where it can be used for non-trivial embedded software applications [5, 15]. However, finding a correctness proof of a system consisting of software running in a physical environment, is often an art, that involves building a model of the software and its physical environment, proving the property of the model, and showing that the model accurately represents the system with respect to this property. (We say that a model M *accurately* represents a system S with respect to a property P if $M \models P$ implies $S \models P$.) We consider systems that consist of a physical and a software part. The software part of the system can, in principle, be modeled automatically [6], and even

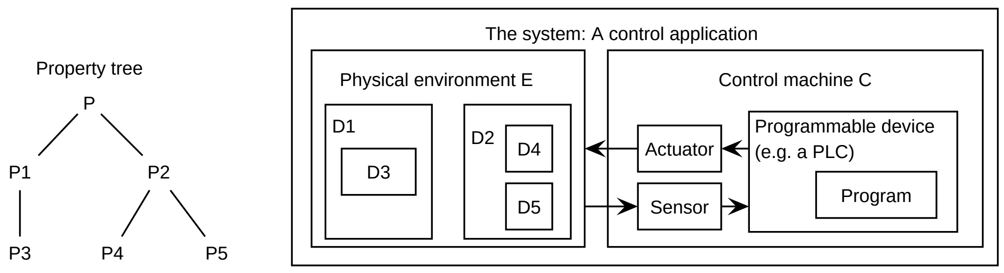

Figure 1: Stepwise decomposition of the environment model and the correctness proof. P is a property of desired behavior D of the environment. D is decomposed into Di and and P_i is a property of D_i.

when this is not done automatically, it is an abstraction operation from one formal object (the program) to another (the model). Modeling the physical part of the system is more difficult, because the modeler must construct and validate a symbolic object, the model, of a nonsymbolic, physical object. If we then prove a property of the model, we need a separate and essentially informal argument that the model shares the property of interest with the system. This process is far from understood, i.e. different people tend to produce different models of the embedded system, and our trust in the models often depends upon our trust in the people who made them. Our aim in this project is to improve the quality of the modeling process that is at the heart of the verification process. By this we mean that

- In the modeling process, reusable and validated knowledge about the modeled system is used and
- The justification that the model accurately represents the system (with respect to a correctness property) is clear and understandable to more people than it now often is.

The first goal makes the modeling process less dependent on the genius of the modeler and the second goal makes the resulting model more reusable.

6.2 Solution idea

Our solution idea is to construct the system model and the proof of its correctness with respect to the property of interest at the same time. We produce the model by means of stepwise decomposition of both the environment and the property of interest, so that the accuracy of the model with respect to the property is justified by a structural similarity between the model and the system. We now elaborate this solution idea in detail.

The systems we consider are control applications (figure 1). They consist of a control machine interacting with a physical environment. The control machine itself consists of a physical part, such as a PLC, sensors and actuators, and a software part. The physical part of the control machine does not concern us here and we will assume in this project that it functions correctly. Initially, we will also assume that all physical entities in the environment function correctly according to a certain specification given to us. For example, in a batch plant producing brine, we assume that a valve receiving the command to open, will actually open [24, 26, 32]. We will later weaken this assumption.

Regardless of the control machine, the environment has certain properties E. For example, in a batch plant for extracting salt from brine, when a valve on a duct opens, gravity will cause fluid to move down through the duct.

Without the control machine, the environment has desirable and undesirable behavior. Connecting the environment to a correct control machine specified by S will cause the environment to engage in certain desirable behavior with desirable properties P. For example, after starting the salt extraction process by evaporation, and after a finite time, one tank will be filled with clean water and another with a concentrated salt solution. Following the approach to requirements engineering proposed by Jackson [13, 17, 31], the correctness problem for the control machine can be formalized as

$$E \wedge S \models P. \tag{1}$$

E describes the possible behavior of the environment (without the control machine) and P describes the desirable behavior of the environment (interacting with the control machine). S is a specification of the control machine. Now, a description of *all* possible environment behavior is usually too large to actually write down, and in any case too large to be used in a realistic verification process. Our method aims, among others, at finding only those few elements of behavior in the environment that we need to verify (1). Secondly, we will choose for P any of the interesting properties that S must make true, rather than taking P to be the conjunction of all of those properties. For example, P may be a liveness property that the controller must make true. This gives us a set of correctness requirements of the form (1), one for each interesting property P. Third, for what follows it is important to note E and P only contain terms referring to the physical environment, and that S only contains terms referring to the interface of shared phenomena between the environment and the control machine.

If we are *designing* the controller, we are given the engineering documentation of the environment and the desired properties P, and need to find environment properties E and a control machine specification S so that (1) is true. If we are *verifying* a given controller, S is given and we still need to prove that (1) is true. Our method can be used in both situations. But note that whether we design S or verify a given S, E is *not* given to us. What is given instead of E is a set of engineering diagrams, such as a P/I diagram showing the physical layout of tanks, tubes, valves, pumps, condensors etc. and other kinds of diagrams. We may also have a set of recipes according to which substances are to be produced, a set of technical specifications of devices in the environment, and possibly other engineering documentation of the environment. The human modeler must find from the very large set of all properties of the environment, those actually needed to prove (1). This is the problem of physical modeling that is the main subject of this project.

Decomposing desired behavior

To find the relevant part E of the very large set of environment properties, we turn to the *desired behavior* D of the environment, that must be brought about by the control machine. This desired behavior by definition satisfies the desired properties, i.e. $D \models P$ for the properties P of interest. We then use the engineering diagrams, recipes, production process descriptions and any other documentation about the environment to decompose the desired behavior into parts, say $D_1, \ldots, D_n$ and we look for a corresponding decomposition of P into $P_1, \ldots, P_n$ such that we can prove that $P_1 \wedge \ldots \wedge P_n \rightarrow P$ and we believe, based on informal, physical arguments, that P_i is true of D_i.

Proving $P_1 \wedge \ldots \wedge P_n \rightarrow P$, however, may not be possible formally unless we add axioms to the proof system, that express certain physical properties of the system, such as an initial condition, a physical decomposition structure, a causal property of the system (e.g. gravity pulls a fluid downwards), or an assumption about a device (e.g. that it functions correctly). The structure of the formal proof is thus

$$A \vdash P_1 \wedge \ldots \wedge P_n \rightarrow P, \tag{2}$$

where A is the set of required axioms about the physical environment needed to prove (2). We call (2) a *property decomposition proof.*

The decomposition of D into $D_1, \ldots, D_n$ is not a design step but a modeling step, where the modeler uses the documentation of the desired behavior of the environment to identify

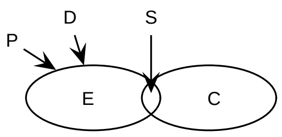

Figure 2: A Venn diagram of all possible phenomena of the control machine C and of the environment E. The specification S of C only refers to shared phenomena. The desired behavior D of E and its desired property P, refer to E and not to any phenomena internal to C. Based on [13].

an existing structure in this behavior. (This desired behavior itself has been designed by the plant engineers, not by the designer or verifier of the control machine.) This is a creative step for the modeler because there are many different structures that can be identified in the environment and the modeler has to choose one. For example, desired behavior of the batch plant may be decomposed following the structure of physical entities such as valves, pumps, tubes etc. or alternatively it may be decomposed following the structure of physical process steps such as heating, cooling and transporting fluid. The modeler must choose a decomposition so that she believes that the control machine can cause P_i to be true of D_i, i.e. the modeler believes that

$$A' \wedge \bigwedge_i D_i \models \bigwedge_i P_i. \tag{3}$$

A' consists of the axioms in A' and any additional axioms needed to prove (3). After possible further augmentation during further decomposition steps, the final set A^* will be our relevant subset of the very large set of all environment properties, i.e. it is the set E we were looking for. So we find E by our need for particular assumptions in the verification of (2) and (3).

As we will see below, after repeated decomposition $\bigwedge_i D_i$ will turn into the specification S. The desired correctness property will then follow if we combine (3) with all the property decomposition proofs (2). Note that proving $A' \wedge D_i \models P_i$ separately for each i is not sufficient. The D_i will interact, so that we must prove (3).

The decomposition process is nonmonotonic, because a step that decomposes D_i into $D_{i_1}, \ldots, D_{i_k}$ and P_i into $P_{i_1}, \ldots, P_{i_k}$ might require us to add axioms to A^j that change the proof that $A^{j+1} \vdash P_1 \wedge \ldots \wedge P_n \rightarrow P$. The additional axioms encode relevant parts of the properties of the physical environment needed by the control machine to enforce property P in the environment.

Further decomposition

A^j, D_i and P_i contain atoms newly introduced by the modeler, whose denotation in the physical world must be defined by the modeler in a dictionary. These definitions are essentially informal but they must unambiguously relate the formulas to the environment; without them, the formulas do not make a statement about the physical environment [17]. The control machine specification S too, used in (1), contains terms that must be defined in the dictionary. They must only refer to the interface of shared phenomena between the environment and the control machine (figure 2). This is the part that can be observed and influenced by the control machine.

In a design situation, if all the D_i refer to the interface of shared phenomena between E and C, then $\bigwedge_i D_i$ is a specification of the controller. It then specifies desired behavior at the interface between the controller and the environment, that jointly with the environment properties A^j will cause P to be true. We can show this using (2) and (3). If not all of the D_i refer to the interface between the control machine and the environment, then we repeat the process of decomposition with each D_i, where the modeler continues to be guided by the engineering documentation of the physical structures she perceives in the environment and by the desired properties P_i.

In a verification situation, S is given and we must find a decomposition of desired environment behavior into this S. Finding this decomposition tree is then an act of understanding why S would bring about D. Once such a tree is found, verification proceeds as explained above.

This approach has been performed in detail, using formal verification tools for the correctness proofs, in one example of designing and verifying a controller for a batch plant [26]. It has the advantage that it uses engineering documentation of the physical environment and the desired property P to guide the modeler in making a model that is accurate with respect to P. The decomposition is guided by both the physical structures found in the environment as well as the structures into which P can be decomposed, so that the fragmentary model A is an accurate representation of the environment with respect to P. This promises reuse of proof fragments for control systems that function in similar environments. Reliance on engineering diagrams promises the use of visual means to increase the understandability of the accuracy argument. This brings us to two groups of research questions.

6.3 Research questions

Control problem frames

The leading question is here: *Can we identify reusable problem frames in the physical environment?* The idea of problem frames (rather than solution patterns) was introduced by Jackson [17, 18] as a means to identify reusable problem structures in different problem domains. Our analysis suggests that we could formalize problem frames for control machines as a property tree (figure 1), an environment model, and formulas of the form (2) and (3). This idea needs to be elaborated and validated. Specific questions to be answered for these frames are:

Q1 How do problem frames compose? The formulas of the form (2) and (3) that come with each applicable problem frame, must be combined to yield the correctness formula for the system of interest. Can we identify and exploit compositionality in the environment to simplify this?

Q2 How can we reuse a property decomposition tree and the correctness proof of a control machine, when we change the environment slightly, for example in a product family?

Q3 How can we reuse the property tree (figure 1) and the correctness proof of a control machine, when we change the desired property slightly?

Q4 We assumed that all devices in the environment function as specified. These assumptions are translated in axioms in A that say how the physical environment behaves. Dropping such an assumption means dropping, or changing, one or more axioms, and causes the controller to be more complex, for it has to deal with a more complex environment. Now, dropping an assumption corresponds to adding a problem concern to a problem frame [18]. Can we classify the simplifying assumptions made about the environment of a control machine, and the concerns that arise when we drop such an assumption?

Model justification

The leading question is here: *Can informal engineering argumentation (physical, graphical, ...) be combined with formal reasoning?* We believe that the decomposition process should provide the justification of the accuracy of the model with respect to the required property. However, the decomposition process cannot be checked by plant engineers unfamiliar with the logic used. In addition, the decomposition process may be constrained by the available formal languages, e.g. the property language of the model checker used to verify correctness claims such as (3). This may influence the accuracy of the decomposition in ways not apparent to the modeler herself, who may read informal meanings into the formulas that are not present from a formal point of view. Visual representations of the formal decomposition that are understandable to the plant engineer may help. Specific questions to be answered are:

Q5 Can we ease the decomposition process by using diagrams? Can we use, for example, domain-specific diagrams understandable to plant engineers? Can we use diagrams familiar to software engineers, such as (parts of) the UML? Can we adapt existing tools to give tool support for this?

Q6 What is the role of the available property languages and specification in the decomposition process? Do they constrain or misguide the decomposition process? Can we exploit their expressiveness to zoom in on relevant properties? Can we compensate for their lack of expressiveness by providing modeling guidelines, e.g. for choosing an appropriate abstraction level? How do we deal with hybrid systems?

Q7 Can we translate the formal correctness proof into a convincing and informal argument as e.g. Meyer [27] does? In our case, the informal argument should be stated in terms of the physics of the situation, that would convince the plant engineers.

Q8 Is there a "control machine physics" that contains frequently used axioms? Such axioms are known from reasoning about change, e.g. an axiom that a state persists until explicitly changed. More generally, how can physical properties be formalized in an intuitive way?

6.4 Research method

We will build up our library of problem frames and justification techniques by doing case studies, starting from the simple and moving to the complex. There are a number of well-known cases from the literature that we will revisit: The sluice control [18], the steam boiler [1], the production cell controller [21], and the elevator controller [31]. In addition, we have done a number of industrial cases that need to be redone using the method of this project: A lacquer production process [4], a smart card personalisation system [25], and a pre-crash airbag controller in a car [3]. Other case studies can be obtained from projects we are currently involved in [2].

We will validate our results in several ways: We will give our library of frames and techniques to verification experts and let them do further cases with it to see if they use the techniques as we intended. Second, we will give our library to software engineers that are not verification experts—i.e. students—to check if they are able to use the problem frames and justification techniques. To avoid getting bogged down in the intricacies of verification tools, this will require verification experts to act as "user friendly verification tools". Third, we will validate our results by applying them to a larger industrial case study. The result of our research will be a library of problem frames relevant for designing and verifying control machines, together with justifications for their accuracy in certain environments.

6.5 Scientific relevance

There is considerable research in algorithms and tools for formal verification, but so far, using these tools is more art than engineering. To make formal verification an industrial possibility, we need to do research into modeling that makes verification accessible to more people than it is now. To our knowledge, the combination of formal verification with problem frames proposed in this project is new. We believe it will advance both the use of problem frames and the applicability of formal verification tools.

6.6 Related research elsewhere

This project aims to elaborate and formalize Jackson's problem frame approach in the domain of control machines [13, 17, 18]. Michael Jackson agreed to work with us in this project. We plan to do at least a number of case studies with him.

Jon Hall did some initial work on formalizing the semantics of problem frames [14]. So far, this has been about formalizing problem frames, not about making formal verification more (re)usable. His proposal to compose and decompose problem frames might be relevant for our research [22].

Decomposition of the property in a tree is reminiscent of goal-oriented requirements engineering methods such as KAOS [8, 19]. Our project proposes a combination with problem frames and formal verification technology not yet proposed in KAOS, and it is not tied to one particular property language, as KAOS is. However, we will use the considerable experience there exists with KAOS to build property trees.

There has been relevant research that identified some frequently occurring patterns in temporal properties [9]. This complements our proposal, because we look for patterns in problem domains, about which properties must be proven.

Existing case studies of formal specification investigate the relationships between specification formalisms rather than trying to identify problem frames [1, 21]. This is relevant for question Q6, where we investigate the influence of the property and specification languages on the property decomposition tree and on the specification of desired behavior.

6.7 Related research of the research team

The IS group has a long history in investigating the combination of formal and informal specification techniques. Informal specification techniques were analyzed and classified in the NYAM framework [30, 31]. An approach to combining formal and informal techniques was presented earlier [29]. Recent results include the formalization of an execution semantics and model checking support for object-oriented statecharts [10] and for UML activity diagrams, and the provision of tools support for model checking activity diagrams (using the nuSMV model checker) [11, 12].

The DIES group investigates systemic aspects of distributed and embedded systems, such as (real) timeliness, security and energy efficiency. We design, implement and analyse experimental embedded systems, yielding insight in both the physical and software parts of the system. We use model checkers to support much of our research. This means that DIES has the expertise to operate as a small, but realistic laboratory for MOCA case studies[4, 7, 16, 20]. MOCA is closely related to the MoMS project of the DIES group [24, 23, 28, 32]. MOCA and MoMS will work in close interaction with each other.

The FMT group has internationally recognized competence in the application of formal methods to modelling and analysis of concurrent systems, in particular embedded and object-oriented systems, and the design and development of related software support tools. Applications include specification, verification, conformance testing, and performance analysis of such systems. Recently developed tools include MOTOR (specification and simulation), TorX (test generation and execution), and ETMCC (model checking of continuous-time Markov chains).

7 Work Program

Phases

The duration of the project is four years. We request two Ph.D. students, working in the IS group (Ph.D. 1) and DIES group (Ph.D. 2), respectively. Ph.D. 1 will work on problem frames and Ph.D. 2 will work on justification. We will work closely with Michael Jackson by doing joint case studies. There is a considerable amount of work to do and it is likely that we will not be able to answer all research questions in four years, even using the slack time in the fourth year. We will start with questions Q1 and Q5 and try to get as far as possible.

Year 1 In the first six months, Ph.D. 1 will familiarize herself with literature about problem frames and goal-oriented requirements engineering (KAOS). Ph.D. 2 will familiarize herself with at least one model checker and one theorem prover. Jointly, they will perform an initial case study with formal verification to become familiar with the technology.

In the next six months, Ph.D. 1 and 2 will jointly perform one or two case studies. Ph.D. 1 will identify problem frames in these (Q1) and Ph.D. 2 will investigate using diagrams to do the decomposition process (Q5).

Year 2 Ph.D. 1 and 2 will perform 2 or 3 additional case studies to elaborate and validate the results of year 1. Ph.D. 1 will then investigate the reuse of property trees and frames in slightly different situations (Q2, Q3). Ph.D. 2 will investigate the influence of property languages and specification languages by using different verification tools for the same problems (Q6).

If there is time left, Ph.D.1 will investigate the effect of dropping environment assumptions on frame concerns (Q4), and Ph.D. 2 will investigate the relation between the formal correctness proofs and the physics of the situation (Q7, Q8). The result of the first two years will be a library of problem frames and model justification techniques for control machines.

Year 3 Ph.D. 1 and 2 will validate the library of problem frames and justification techniques, first on verification experts in our own groups (and the group of formal methods), and then on groups of software engineering students. The resulting library will be further validated by performing an industrial case study.

Year 4 The PhD students will complete ongoing research, write their thesis and prepare the defence.

Educational aspects

The PhD students will take part in relevant courses of SIKS and IPA about modeling and verification technology. They will take part in courses provided by the UT on how to organize one's research, write research papers, and give presentations.

8 Expected Use of Instrumentation

The University will provide computing equipment and daily supervision for the project members.

9 Literature

Five Main Publications of the Research Team

[1] E. Brinksma and A. Mader. Verification and optimization of a PLC control schedule. In K. Havelund, J. Penix, and W. Visser, editors, *7th Int. SPIN Workshop on Model Checking of Software*, volume LNCS 1885, pages 73–92, Stanford Univ., California, Aug 2000. Springer-Verlag, Berlin. `http://spinroot.com/spin/Workshops/ws00/18850075.pdf`.

[2] A. Mader, E. Brinksma, H. Wupper, and N. Bauer. Design of a PLC control program for a batch plant - VHS case study 1. *European Journal of Control*, 7(4):416–439, 2001. `http://www.cs.utwente.nl/~mader/PAPERS/REPORT.ps.gz`.

[3] R. Eshuis and R.J. Wieringa. Tool support for verifying UML activity diagrams. *IEEE Transactions on Software Engineering*, 30(7):437–447, July 2004.

[4] R.J. Wieringa. A survey of structured and object-oriented software specification methods and techniques. *ACM Computing Surveys*, 30(4):459–527, December 1998.

[5] R.J. Wieringa. *Design Methods for Reactive Systems: Yourdon, Statemate and the UML.* Morgan Kaufmann, 2003.

References

[1] J.-R. Abrial, E. Börger, and H. Langmaack, editors. *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control.* Springer, 1996. Lecture Notes in Computer Science 1165.

[2] AMETIST. Advanced Methods for Timed Systems, Esprit-LTR project IST-2001-35304. `http://ametist.cs.utwente.nl`.

[3] AMETIST. Case study 3: Service allocation. `http://wwwhome.cs.utwente.nl/~mader/PROJECTS/MoMS/`.

[4] H. C. Bohnenkamp, H. Hermanns, R. Klaren, A. Mader, and Y. S. Usenko. Synthesis and stochastic assessment of schedules for lacquer production. In *1st Int. Conf. on Quantitative Evaluation of Systems (QEST)*, page to appear, Enschede, The netherlands, Sep 2004. IEEE Computer Society Press, Los Alamitos, California. `http://www.ub.utwente.nl/webdocs/ctit/1/00000101.pdf`.

[5] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L. J. Hwang. Symbolic model checking: `10^20` states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[6] J. C. Corbett, M. B. Dwyer, J. Hatcliff, S. Laubach, C. S. Pasareanu, Robby, and H. Zheng. Bandera: extracting finite-state models from Java source code. In *22nd Int. Conf. on on Software engineering*, pages 439–448, Limerick Ireland, Jun 2000. ACM, New York.

[7] R. Corin, S. Etalle, P. H. Hartel, and A. Mader. Timed model checking of security protocols. In *2nd ACM Workshop on Formal Methods in Security Engineering: From Specifications to Code (FMSE)*, page to appear, Washington D. C., Oct 2004. ACM Press, New York.

[8] R. Darimont and A. van Lamsweerde. Formal refinement patterns for goal-driven requirements elaboration. In *Fourth ACM Symposium on the Foundations of Software Engineering (FSE4)*, pages 179–190, 1996.

[9] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the 1999 international conference on software engineering : ICSE*, pages 411–420, New York, N. Y., May 1999. ACM.

[10] R. Eshuis, D.N. Jansen, and R.J. Wieringa. Requirements-level semantics and model checking of object-oriented statecharts. *Requirements Engineering Journal*, 7(4):243–263, 2002.

[11] R. Eshuis and R.J. Wieringa. Verification support for workflow design with UML activity graphs. In *24th International Conference on Software Engineering (ICSE 2002)*, pages 166–176, 2002.

[12] R. Eshuis and R.J. Wieringa. Tool support for verifying UML activity diagrams. *IEEE Transactions on Software Engineering*, 30(7):437–447, July 2004.

[13] C.A. Gunter, E.L. Gunter, M.A. Jackson, and P. Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3):37–43, May/June 2000.

[14] J. G. Hall and L. Rapanotti. A Reference Model for Requirements *Engineering*. In *Proceedings of the 11th Joint International Conference of Requirements Engineering*, pages 181–187. IEEE Computer Science Press, 2003.

[15] J. Y. Halpern, R. Harper, N. Immerman, P. G. Kolaitis, M. Vardi, and V. Vianu. On the unusual effectiveness of logic in computer science. *The Bulletin of Symbolic Logic*, 7(2):213–236, 2001.

[16] P. H. Hartel, P. van Eck, S. Etalle, and R. J. Wieringa. Modelling mobility aspects of security policies. In *Construction and Analysis of Safe, Secure and Interoperable Smart cards (CASSIS)*, page to appear, Marseille, France, Mar 2004. Springer-Verlag, Berlin.

[17] M.A. Jackson. *Software Requirements and Specifications: A lexicon of practice, principles and prejudices.* Addison-Wesley, 1995.

[18] M.A. Jackson. *Problem Frames: Analysing and Structuring Software Development Problems.* Addison-Wesley, 2000.

[19] A. van Lamsweerde, R. Darimont, and E. Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, 24(11):908–926, November 1998.

[20] Y. W. Law, R. Corin, S. Etalle, and P. H. Hartel. A formally verified decentralized key management architecture for wireless sensor networks. In M. Conti, S. Giordano, E. Gregori, and S. Olariu, editors, *4th IFIP TC6/WG6.8 Int. Conf on Personal Wireless Communications (PWC)*, volume LNCS 2775, pages 27–39, Venice, Italy, Sep 2003. Springer-Verlag, Berlin.

[21] K. Lewerentz and T. Lindner, editors. *Case Study Production Cell — A Comparative Study in Formal Software Development.* Springer, 1994. Lecture Notes in Computer Science.

[22] Z. Li, J. G. Hall, and L. Rapanotti. Reasoning about decomposing and recomposing problem frames: a case study. In *Proceedings of 1st International Workshop on Applications and Advances of Problem Frames*, pages 49–53. The Institution of Electrical Engineers, 2004.

[23] A. Mader. A classification of PLC models and applications. In R. Boel and G. Stremersch, editors, *5th Int. Workshop on Discrete Event Systems (WODES) – Discrete Event Systems, Analysis and Control*, pages 239–247, Ghent, Belgium, Aug 2000. Kluwer Academic Publishers, Massachusetts. `http://www.cs.utwente.nl/~mader/PAPERS/modelbox.ps`.

[24] A. Mader. What is the method in applying formal methods to PLC applications? In S. Engel, S. Kowalewski, and J. Zaytoon, editors, *4th Int. Conf. Automation of Mixed Processes: Hybrid Dynamic Systems (ADPM)*, pages 165–171, Dortmund, Germany, 2000. Shaker Verlag, Aachen, Germany. `http://www.cs.utwente.nl/~mader/PAPERS/method.ps.gz`.

[25] A. Mader. Deriving schedules for a smart card personalisation system. Technical report TR-CTIT-04-05, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, Jan 2004. `http://www.ub.utwente.nl/webdocs/ctit/1/000000e8.pdf`.

[26] A. Mader, E. Brinksma, H. Wupper, and N. Bauer. Design of a PLC control program for a batch plant - VHS case study 1. *European Journal of Control*, 7(4):416–439, 2001. `http://www.cs.utwente.nl/~mader/PAPERS/REPORT.ps.gz`.

[27] B. Meyer. On formalism in specifications. *IEEE Software*, pages 6–26, January 1985.

[28] MoMS. Methods of modeling embedded systems. `http://wwwhome.cs.utwente.nl/~mader/PROJECTS/MoMS/`. NWO Project 632.001.202.

[29] R.J. Wieringa. LCM 3.0: Specification of a control system using dynamic logic and process algebra. In C. Lewerentz and T. Lindner, editors, *Formal Development of Reactive Systems — Case Study Production Cell*, pages 333–355. Springer, 1994. Lecture Notes in Computer Science 891. ftp://ftp.cs.utwente.nl/pub/doc/MAICS/94-ProductionCell.ps.Z.

[30] R.J. Wieringa. A survey of structured and object-oriented software specification methods and techniques. *ACM Computing Surveys*, 30(4):459–527, December 1998.

[31] R.J. Wieringa. *Design Methods for Reactive Systems: Yourdon, Statemate and the UML*. Morgan Kaufmann, 2003.

[32] H. Wupper and A. Mader. System design as a creative mathematical activity. Technical report CSI-R9919, Univ. of Nijmegen, 1999. `http://www.cs.kun.nl/research/reports/full/CSI-R9919.ps.Z`.