Project no.: **ICT-FP7-STREP-214755**

Project full title: **Quantitative System Properties in Model-Driven Design**

Project Acronym: **QUASIMODO**

Deliverable no.: D1.3

Title of Deliverable: Model Process Improvement

Contractual Date of Delivery to the CEC:	**Month 24**
Actual Date of Delivery to the CEC:	**Month 24 (February 10, 2010)**
Organisation name of lead contractor for this deliverable:	**P02 ESI(RU)**
Author(s):	**Frits Vaandrager**
Participant(s):	**all**
Work package contributing to the deliverable:	**WP 1**
Nature:	**R**
Version:	**1.0**
Total number of pages:	**10**
Start date of project:	1 Jan. 2008 **Duration:** 36 month

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)

Dissemination Level

PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract:

This deliverable describes the view of the QUASIMODO project on model process improvement, and the lessons we learned concerning the modelling process by doing the QUASIMODO case studies.

Keyword list: quality of models, model process improvement.

Contents

1 Introduction

This deliverable describes the view of the QUASIMODO project on model process improvement, and the lessons we learned concerning the modelling process by doing the QUASIMODO case studies.

Eykhoff [5] defined a *mathematical model* as a "representation of the essential aspects of an existing system (or a system to be constructed) which presents knowledge of that system in usable form". Mathematical models can take many forms, including dynamical systems, statistical systems or differential equations. Within QUASIMODO, we study certain specific types of discrete event dynamical systems, namely timed, probabilistic and priced automata.

During the last two decades, research on timed, probabilistic and priced automata focussed on theory and algorithms for efficient exploration of large state-spaces. This research has been quite successful, and by now model-based verification technology has reached the maturity in which it can be (and has been) applied to many non-trivial embedded systems applications. However, as pointed out by [3], "current research seems to take the construction of verification models more or less for granted, although their development typically requires a coordinated integration of the experience, intuition and creativity of verification and domain experts. There is a great need for systematic methods for the construction of verification models to move on, and leave the current stage that can be characterized as that of *model hacking*. The ad-hoc construction of verification models obscures the relationship between models and the systems that they represent, and undermines the reliability and relevance of the verification results that are obtained." Another reason why we need to pay more attention to the construction of models is that this is an excellent way to find more bugs. Our experience is that one finds more bugs during the careful construction of models than during the subsequent model checking or testing phase. In a case study where we applied Uppaal to model and analyze the Zeroconf protocol [2], we found six places where RFC 3827 [4] is incomplete/unclear. All of these six mistakes/ambiguities were found during the modeling phase. In the QUASIMODO Chess case study, we spend several months to analyze a Uppaal model of the gMAC clock synchronization algorithm but did not find any flaw [6]. Only after we decided to invest in a more accurate modeling of the algorithm, we quickly discovered a flaw in the current implementation [8].

In the next section we identify seven criteria which we think a good model should satisfy. We think that systematically checking and documenting these criteria will help to structure the modeling process, and will lead to better models. Section 3 discusses in more detail some of the QUASIMODO case studies and the lessons we learned from them concerning the modelling process.

2 What is a Good Model?

To some extent, building good models is an art. Dijkstra's motto "Beauty is our business" applies to models as well as to programs. Nevertheless, we can state seven criteria for good models. These criteria are in some sense obvious, and any person with experience in modelling will often try to adhere to them. But surprisingly our list of criteria has - to the best of our knowledge - not been described elsewhere in the literature, although most of them occur in a technical report of Mader, Wupper and Boon [7].[1] Often, the criteria are hard to meet and typically several of them are conflicting. In practice, a good model is often one which constitutes the best possible compromise, given the current state-of-the-art of tools for modelling and analysis. But a truly beautiful model meets all the criteria! We refer to [7] for further links to related work in the areas of software engineering, requirements analysis, and design.

1. A good model has a clearly specified **object of modelling**, that is, it is clear what thing the model describes. The object of modelling can be (a part of) an existing artefact or physical system, but it may also be a document that informally specifies a system or class of systems (for instance a protocol standard), and it may even be a collection of ideas of a design team about a system they construct, expressed orally and/or by some drawings on a whiteboard.

2. A good model has a clearly specified **purpose** and (ideally) contributes to the realization of that purpose. Possible purposes include: communication between stake holders, verification of specific properties (safety, liveness, timing,..), analysis and design space exploration, code generation, and test generation. A model can be descriptive or prescriptive. If a model has to serve several distinct purposes then often it is better to construct multiple models rather than one.

3. A good model is **traceable**: each structural element of a model either (1) corresponds to an aspect of the object of modelling, or (2) encodes some implicit domain knowledge, or (3) encodes some additional assumption. Additional assumptions are for instance required when a protocol standard is incomplete (e.g., it does not specify how to handle certain events in certain cases). Links between the structural elements of the model and the aspects of the object of modelling should be clearly documented. A distinction must always be made between properties of (a component of) a model and assumptions about the behavior of its environment.

4. A good model is **truthful**: relevant properties of the model should also carry over to (hold for) the object of modelling. Typically, for each (relevant) behavior of the object of modelling there should be a corresponding behavior of the model, and/or for each behavior of the model there should be a corresponding behavior of the artefact. In the construction of models often idealizations or simplifications are necessary in order to allow for the use of a certain modeling formalism or in order to be able to analyze the model. In these cases, the

[1] We see this as a clear indication of the lack of interest for the methodology of modeling in our field.

model may not be entirely truthful. The modeller should always be explicit about such idealizations/simplifications, and have an argument why the properties of the idealized model still say something about the artefact. In the case of quantitative models this argument will typically involve some error margin. In the case of nondeterministic models it frequently occurs that a model "overapproximates" reality, and that certain behaviors that are possible in the model are not possible for the artefact.

5. A good model is **simple** (but not too simple). Occam's razor is a principle particularly relevant to modelling: among models with roughly equal predictive power, the simplest one is the most desirable. Hence, the number of states and state variables should be as small as possible, and the level of atomicity of transitions should be as coarse grained as possible (but not coarser), i.e., the number of transitions should be minimal given the intended use of the model. Preferably, things should be written only once, and one should avoid ugly encodings. Preferably, the model uses stable, well-defined and well-understood concepts and semantics.

6. A good model is **extensible and reusable**, that is, it has been designed to evolve and be used beyond its original purpose. Typically, if one defines models in a modular and parametric way this allows for dimensioning, future extensions and modifications, especially if modules have well-defined interfaces. Ideally, a model should not just describe the specific system at hand: by appropriate instantiation and dimensioning it should be possible to model a whole class of similar systems.

7. A good model has been designed and encoded for **interoperability and sharing** of semantics. Model-driven development of an embedded system typically leads to a plethora of models, all presenting different views on and abstractions of the system. If a model is not somehow linked to other models, its usefulness will be limited. Ideally therefore, the relationships between all models should be properly defined, for instance via formal refinement relations.

Clearly, there are many relationships and dependencies between the criteria. If a model is traceble, that is, links between the structural elements of the model and the aspects of the object of modelling are clearly documented, then chances increase that the model will be thrutful. Also, if a model has been set up in a modular way, then one may apply a divide-and-conquer strategy both for establishing truthfulness of the model and for analysis. Etc, etc.

We think that the above criteria may help engineers to construct good models: just by systematically checking and documenting that all criteria are met, and by understanding and documenting the various tradeoffs during the modeling phase, the quality of models will significantly increase. We also think that our criteria may help to define a roadmap for research: our current tools and techniques often simply do not allow engineers to build good models (e.g., due to state space explosion a verification purpose cannot be achieved, the syntax is not sufficiently expressive to allow for simple models, and it is not possible to relate models).

3 Case Studies

Within the QUASIMODO project, we paid special attention to the modelling process in a number of case studies, that we will discuss below.

3.1 Zeroconf Case Study

Participants Jasper Berendsen and Frits Vaandrager (RU), together with Biniam Gebremichael and Miaomiao Zhang.

Results The model checker Uppaal is used to formally model and analyze parts of Zeroconf, a protocol for dynamic configuration of IPv4 link-local addresses that has been defined in RFC 3927 of the IETF. Our goal has been to construct a model that (a) is easy to understand by engineers, (b) comes as close as possible to the informal text (for each transition in the model there should be a corresponding piece of text in the RFC), and (c) may serve as a basis for formal verification. Our modeling efforts revealed several errors (or at least ambiguities) in the RFC that no one else spotted before. We present two proofs of the mutual exclusion property for Zeroconf (for an arbitrary number of hosts and IP addresses): a manual, operational proof, and a proof that combines model checking with the application of a new abstraction relation that is compositional with respect to committed locations. The model checking problem has been solved using Uppaal and the abstractions have been checked by hand. [2]

Lessons learned Within this case study, we looked specifically at the issues of traceability and faithfullness of the model. The most important lesson that we learned is that this pays off: six flaws/ambiguities were discovered that were overlooked by the protocol designers who wrote the RFC, and by us in our earlier modelling efforts.

If one constructs a model that is traceable and faithful, then often it will be too detailled for direct verification using a model checker. We deviced a compositional abstraction technique for transforming the original model in a more abstract tractable model. We proved correctness of the abstraction by hand. Clearly, there is a great need for mechnanized support of these abstractions in Uppaal. Currently, Uppaal also lacks a clear notion of component/module.

3.2 Chess Case Study

Participants Faranak Heidarian, Frits Vaandrager, Mathijs Schuts, and Feng Zhu (RU)

Results We present a detailled timed automata model of the clock synchronization algorithm that is currently being used in a wireless sensor network (WSN) that has been developed by the Dutch company Chess. Using the Uppaal model checker, we establish that in certain cases a static, fully synchronized network may eventually become unsynchronized if the current algorithm is used, even in a setting with infenitesimal clock drifts. [8]

Lessons learned Model checking projects are often carried out a posteriori: the artefact exists and has been documented in some manual of protocol standard. An advantage of such projects is that the object of modelling is clear. A disadvantage may be that the potential impact of the work is limited. The Chess case study is an example of a situation in which timed automata technology is applied during the design of a new system, and may actually impact this design. In such case studies, the object of modelling (a prototype, or just ideas of the engineers) is a moving target, which changes every week or sometimes every day. In order to keep up with the design team, it is essential to have frequent meetings between the modeler and the designers. Preferably, the modeler should be part of the design team and work at the same location. Our mistake was that we were too late in realizing this. As a consequence, our first Uppaal model of the protocol [6] was not faithful. Interestingly, since the actual implementation may become unsynchronized, the solution proposed in [6] may be of practical interest (although certainly some changes are required).

3.3 Océ case study

Participants Georgeta Igna, Frits Vaandrager, Israa AlAttili, Fred Houben, Steffen Michels, Feng Zhu (RU), Jacob Illum and Kim Larsen (AAU)

Results The data path of a printer/copier encompasses the complete path of the image data (the bit stream) from source (for example the network) to target (the imaging unit). In order to reach Océ's objective of genuine system adaptability, also the data path has to be adaptive because its properties heavily influence the image quality of the end product as well as system behavior aspects that have an eminent effect on usability. At run-time changes in the environment (or in the observed image quality, using a feedback mechanism) may for instance require the use of different algorithms in the data path, deadlines for completion of computations may change, new jobs may suddenly arrive, and resource availability may change. To realize this type of behavior in a predictable way is a major challenge. Currently it is already impossible to quickly evaluate cost, energy, performance aspects of various data path implementation solutions at design-time. This does not only hold for changing, adaptive functionality (steered by load, content, print quality), but even for a given fixed functionality. Partner ESI/RU is involved in a project (named Octopus) with Océ in which Uppaal is used to make detailled models of the datapath of printer/copiers and to analyze their behavior. Due to their complexity, these models provide an excellent challenge for the new analysis and synthesis techniques that are being developed within Quasimodo.

Georgeta Igna spent a lot of time at Océ to construct - in close interaction with the designers - detailled Uppaal models of a new machine that is currently being developed, and to use these models for design space exploration. A publication describing these results is currently being reviewed at Océ and will be submitted shortly.

In [1], we applied Uppaal Tiga to automatically compute adaptive scheduling strategies for a simplified version of the model. As far as we know, this is the first application of timed automata technology to an industrial scheduling problem with uncertainty in job arrivals.

Lessons learned In principle, Uppaal is able to faithfully model the datapath of realistic printer designs. State space explosion is clearly an issue, but can be kept under control by also including (some of) the scheduling rules used by Océ within the model. One technical issue that we faced is that although essentially the behavior of the model is fully deterministic when all the scheduling rules are added, the resulting Uppaal model is not (and suffers from state space explosion) due to interleavings of internal actions of the various components. We resolved this by using the channel and process priorities from Uppaal, but a better solution would be to extend Uppaal with support for confluence detection and/or partial order reduction.

A lesson we learned is that it is extremely difficult to maintain correctness of the model in a setting where the object of modelling has such a high complexity. There is not a single document describing the design. In fact there is not a single person who is able to answer all the questions that need to be answered in order to obtain a good model: the knowledge is spread over a large design team. For the engineers at Océ it is difficult to understand the intricacies of the Uppaal model. The syntax of Uppaal is not sufficiently expressive to describe the design in such a way that a small change in the design corresponds to a small change in the model. Due to these difficulties, we decided to develop a high level language for describing the designs, together with a translation to Uppaal: one the one hand this will make it much easier to communicate with the engineers, and on the other hand it will reduce the chances of introducing errors in the Uppaal model.

3.4 Scooter case study

Participants Jiansheng Xing (UT)

Results Chessway has designed and realized successfully its first generation of a self balancing scooter. However, there still exist many problems to be solved for improving the scooter. For the next generation, the most important challenges can be classified into two categories: 1. the design should include formally defined system states such that safety measurement and power management is easier to be implemented. 2. Verification of the proposed solution during design phase and generation of code and test cases from the verified design. Confronted with such challenges, formal method are a promising approach to come to a correct solution. UPPAAL, being a popular model checking tool, is our first choice.

Using UPPAAL, we have formally defined system states such as Charging, PowerDown, Hibernate, Checking, and Drive. Also, we have introduced safety signals such as warning and unsafe into this model. With these states and signals, we can verify that the system satisfy some safety properties or functional specifications. For example, we know that the battery will not be over-consumed; if we know the energy consumption, the remaining running time of the scooter can be calculated. Next, we anticipate power management policy can be included such that performance of different policies can be analyzed and compared. And at the same time, we are working on the generation of code and test cases from the verified model.

In the modeling of scooter system, some techniques have been introduced to improve or simply the modeling process. The interaction of scooter with user is abstracted and implemented as

a timed automaton. This abstraction is very powerful as it can model all kinds of inputs from user and it greatly simplifies the modeling process. This technique applies to simple interactions where the user has few statuses and it can be seen as a test case generation engine for the model itself. Unsafe timed automaton introduces a technique for nondeterministic triggering of failure signals which makes the modeling more succinct. This technique can be enhanced by introducing an urgent channel which guarantees that the triggering of signals will occur. For the implementation of distributed control, the channels and related status variables are declared parameters of the scooter model, and then we can easily specify two or more distributed scooter controllers using this model. Obviously, one controller setting is much simpler to be created and then easily extended to more complicated distributed applications. This template feature is supported by UPPAAL and is fully utilized here for the simplification of the modeling process. With this UPPAAL model, we can analyze the timing performance of battery usage by labeling all non-energy-consuming status with urgent. This abstraction can also be generalized such that we can analyze a specific performance measure (energy usage) by establishing a linear relationship with time. With this technique, we can analyze a specific measure with a simple model rather than a more complicated model (for instance a UPPAAL Cora model).

Lessons Learned This case study nicely illustrates the power of nondeterminism in modeling. Nondeterminism (as supported by Uppaal) allows one to define simple, abstract models of systems. The behavior of the system is in a sense “overapproximated”, but this is ok as long as the purpose of the model is to establish safety properties.

4 Future Work

Our plan is to our seven criteria in detail for each of the case studies that will be presented in the QUASIMODO handbook.

Bibliography

[1] I. AlAttili, F. Houben, G. Igna, S. Michels, F. Zhu, and F. W. Vaandrager. Adaptive scheduling of data paths using Uppaal Tiga. In S. Andova et.al, editor, *Proceedings First Workshop on Quantitative Formal Methods: Theory and Applications (QFM’09)*, volume 13 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–12, 2009.

[2] J. Berendsen, B. Gebremichael, F. W. Vaandrager, and M. Zhang. Formal specification and analysis of zeroconf using uppaal. *ACM Transactions on Embedded Computing Systems*, 2010. To appear.

[3] E. Brinksma and A. Mader. On verification modelling of embedded systems. Technical Report TR-CTIT-04-03, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, January 2004.

[4] S. Cheshire, B. Aboba, and E. Guttman. Dynamic configuration of IPv4 link-local addresses (RFC 3927), May 2005. `http://www.ietf.org/rfc/rfc3927.txt`.

[5] P. Eykhoff. *System Identification: Parameter and State Estimation*. Wiley & Sons, 1974.

[6] F. Heidarian, J. Schmaltz, and F. W. Vaandrager. Analysis of a clock synchronization protocol for wireless sensor networks. In A. Cavalcanti and D. Dams, editors, *Proceedings 16th International Symposium of Formal Methods (FM2009), Eindhoven, the Netherlands, November 2-6, 2009*, volume 5850 of *Lecture Notes in Computer Science*, pages 516–531. Springer, 2009.

[7] A. Mader, H. Wupper, and M. Boon. The construction of verification models for embedded systems. Technical Report TR-CTIT-07-02, Centre for Telematics and Information Technology, University of Twente, The Netherlands, 2007.

[8] M. Schuts, F. Zhu, F. Heidarian, and F. W. Vaandrager. Modelling clock synchronization in the Chess gMAC WSN protocol. In S. Andova et.al, editor, *Proceedings Workshop on Quantitative Formal Methods: Theory and Applications (QFM'09)*, volume 13 of *Electronic Proceedings in Theoretical Computer Science*, pages 41–54, 2009.