On the Roles of Functions and Objects in System Specification

Axel Jantsch
Royal Institute of Technology
Electrum 229
SE-16440 Kista, Sweden
Email: axel@ele.kth.se

Ingo Sander
Royal Institute of Technology
Electrum 229
SE-16440 Kista, Sweden
Email: ingo@ele.kth.se

ABSTRACT

We present an analysis of the benefits and drawbacks of function and object based models in system specification. Functional models should be used for functional design space exploration and as a functional reference model throughout all design and validation activities. Object based models should be used for architectural design space exploration and as a design specification for the following design and implementation phases. Thus, the question is not which one to adopt in system specification, but how to integrate them. We argue that the integration should be based on an explicit formulation of design decisions with a tool handling the consequences of the decisions. In this way a functional model can be transformed into an object based model efficiently and systematically and a discontinuity in the design process is avoided. We consider it important that the question of benefits of functional and object based models is decided by means of experiments. To this end we propose an experiment that would confirm or falsify our hypothesis.

1. INTRODUCTION

A system specification serves a dual purpose.

❶ **Application oriented:** *It is a means to study if the proposed system functionality will indeed be a solution to the posed problem with all its functional and non-functional requirements and constraints, i.e. to make sure to make the right system.*

❷ **Implementation oriented:** *It defines the functionality and the constraints of the system and is a base for the following design and implementation phases.*

The first purpose is directed towards the problem domain and the concepts and terms should reflect the application area. For example, if we want to specify an ATM switch with some operation and maintenance functionality, we should model the virtual path (VP) and virtual channel (VC) switching of ATM cells and the VP loopback functionality. The system functions should be expressed in the terminology of the application domain and it should be modelled in terms of the observable behaviour at the interfaces. This allows the exploration of the system functionality in order to find the appropriate set of system functions that would fulfil the requirements. It also facilitates the analysis and discussion with domain experts.

The second purpose is directed towards implementation for which the specification should be a starting point and a reference. To assess if a feasible implementation can be found for a given functionality, implementation and technology aspects need to be considered. To do this with sufficient accuracy, system level designers usually assume a structural partitioning and an approximate assignment of functions to components. Such an architecture facilitates the estimation of implementation properties such as cost, size, power consumption, and performance with greater accuracy than a pure functional model can provide.

It is easy to see that the application oriented purpose is best met with a pure functional model, which captures the system functionality without implying a system internal structure. This has been argued convincingly many times, e.g. in [2, 4, 7, 9]. It is also apparent from research in requirements engineering [21] where scenarios, which resemble system functions, play an important role [19].

On the other hand for the implementation oriented purpose a pure functional model is not very appropriate. An architectural model which reflects a partitioning into parallel activities and objects would serve this need better, because this allows the identification of implementation components such as processors, memories, functional units, communication structures, etc. The proliferation of proposals to use a parallel process based notation or object oriented languages for system specification [3, 5, 7, 20] is motivated by this architecture oriented point of view. In the following we use the term “object based models” to denote models described in terms of concurrently active objects or processes. Thus, it includes models written in SDL, VHDL, Erlang, SpecChart and concurrent extensions of C and C++.

In this paper we argue that both, a function based model (FBM) and an object based model (OBM) is required for complex system development, and that ignoring one of them will lead to inefficient development and inferior design.

2. FUNCTIONS VERSUS OBJECTS

The dichotomy of functions and objects or functionality and structure is a fundamental one in the area of electronic design. A.C. Sodan [17] discussed various dichotomies in computer science with the conclusion that significant progress has always been made when the fruitful integration of a dual pair was achieved.

2.1 Objects in System Specification

At the system level an object based model, reflecting the architecture of the system is necessary to structure the design process and assign different parts and responsibilities to different designers. It is also essential to make reasonably accurate estimations about implementation properties such as performance, size, cost, and power consumption. However, every object based model contains many design decisions and reduces the design space significantly. As illustrating example we use a network terminal system which has been modeled based on a set of requirements from ITU-T and Ericsson Infocom Systems [8]. The objective of this activity was to

develop a system level model in SDL. SDL [12] is a language based on concurrent processes and finite state machines. The architecture selected in this effort is shown in figure 1. At the top level four blocks

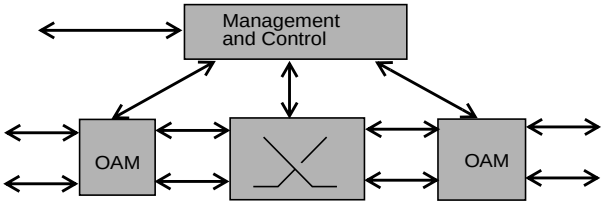

Figure 1. Separate management, OAM, and switch

are distinguished and modeled with separate processes. The OAM blocks realize operation and maintenance functionality performed on the ATM cell streams before and after they are processed by the switch. The management block is responsible for setting-up and releasing connections and for controlling OAM functions. Although the intention was to avoid design decisions as far as possible, several decisions had to be taken in order to model it in SDL. Some of them have far reaching consequences. The most important of these was the partitioning into four parallel processes. While this partitioning seemed very natural when given the requirements, several alternative architectures are reasonable and might in fact be preferable for a particular product with certain cost and performance requirements. If we consider briefly a possible HW/SW partitioning we find out, that the architecture in figure 1 is not a good starting point for various reasonable partitionings. The switch and the interfaces for the ATM cell streams operate with high data rates and the OAM and management functions can operate at a much slower rate. Such a

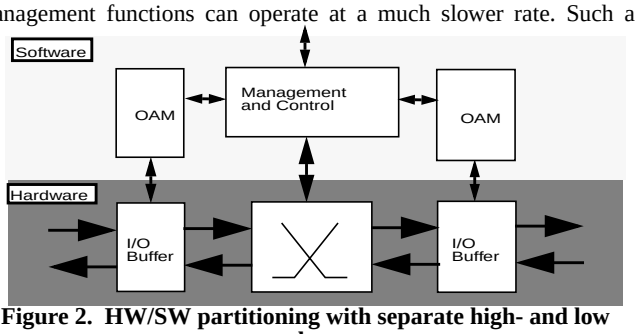

Figure 2. HW/SW partitioning with separate high- and low speed parts

consideration might lead to an architecture outlined in figure 2, where the OAM blocks of figure 1 are split into a high-speed and a low-speed part. After this transformation it is not clear, if the OAM blocks and the management and control block should remain separate processes. Since all of them will be implemented in software, this question will depend on the selected real-time operating system and the given real-time constraints. It is reasonable to assume that the two OAM blocks in figure 2 will be merged, because they represent an identical functionality operating on different streams.

A merging of the OAM blocks of figure 1 might be reasonable even if they are implemented in hardware together with the switch. This would result in an architecture shown in figure 3.

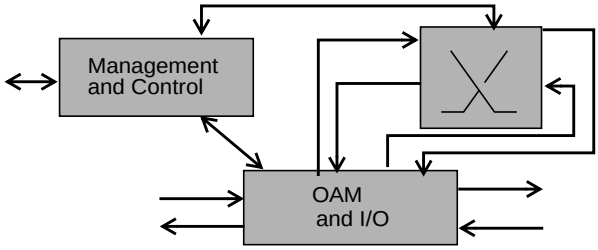

Figure 3. Architecture with a shared OAM block

An alternative solution for a low cost version of the product may be a pure software implementation. In that case we might go for only one execution thread without parallel processes to avoid the overhead of a complex operating system. Hence, an architecture as shown in figure 4 may be most appropriate.

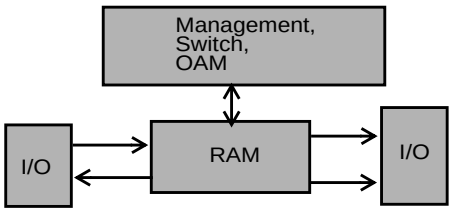

Figure 4. All-SW architecture

We conclude from these considerations that any SDL model of such a system would determine a specific architecture which in turn implies certain design decisions and specific trade-offs. An SDL model cannot be written in an architecture independent way because it is based on concurrent processes communicating by means of conceptually complex protocols. To avoid concurrent processes in SDL would mean to use only a flat finite state machine model without partitioning and hierarchy, which is clearly not feasible for complex and large systems. The same argument holds for many similar languages based on communicating concurrent processes such as VHDL, CFSM, SystemC, etc.

Hence, once the system has been modelled in terms of concurrent processes, it is not suitable any more for purpose 1 of the specification document, i.e. to study and explore the system functionality, because it is very difficult to add, modify and delete system functions since each such change might require a substantial change in the architecture. Assume for instance we have started out with the architecture of figure 2. Then we want to investigate the addition of a certain performance monitoring function. This function would scan every ATM cell, check its integrity and count the number of errors observed. If the number of errors per time unit on a specific virtual connection exceeds a threshold value, error location procedures would be invoked and the network management system would be informed. From this description it is reasonable to assume, that part of this functionality would be located at the I/O Buffers (checking the integrity of ATM cells), another part in the OAM blocks (maintaining the state of each virtual connection and invoking and coordinating the error location procedures), and a third part in the Management and Control block (informing the network management system, receiving orders from management system and coordinating the following actions). It is clear that all these blocks must be changed, the interfaces between these blocks must perhaps be adapted, and the entire remaining functionality of the system must be validated again due to the risk of introduced errors. Moreover, a possible consequence might be that the overall architecture must be changed because another architecture might match the new functionality better. Perhaps part or all of the OAM functions should be moved into the I/O buffers to simplify or eliminate the interface between the OAM and the I/O blocks. So many modifications with the high probability to introduce errors make an efficient exploration at the level of system functions infeasible. At the functional design level an object based model promotes a code-and-fix method rather than a systematic design space exploration.

However, such a description based on concurrent processes serves purpose 2 of a specification document very well, because it allows to identify implementation objects. Consequently it allows to estimate implementation properties with sufficient accuracy.

2.2 Functions in System Specification

A functional description of the network terminal system derived from

the same requirements definitions is shown in figure 5. The function

```
nt (i1, i2, mi) = (o1, o2, mo)
where   (o2, s4, s10) = oam (i2, s3, s9)
        (s3, s1, s5) = switch (s2, s4, s6)
        (o1, s2, s7) = oam (i1, s1, s8)
        (mo, s8, s5, s9) = management (mi, s7, s5, s10)
```

Figure 5. Functional specification

`nt` is composed of the functions `oam`, `switch`, and `management`. This can also be graphically represented as shown in figure 6. Although this seems to be similar to figure 1, the symbols

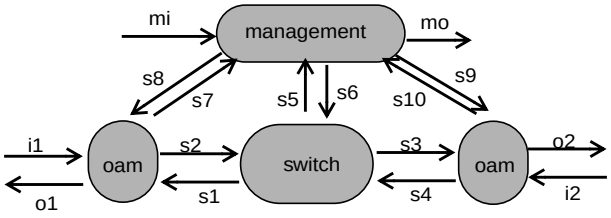

Figure 6. A functional description graphically represented.

have to be interpreted differently. The boxes in figure 6 represent side effect free functions and the arcs represent simple parameter passing. The outer interfaces of function `nt` represent streams of ATM cells (`i1, i2, o1, o2`) and streams of communication messages (`mi, mo`) with the management system. So the fundamental difference between figure 1 and figure 6 is, that the communication in figure 1 is a complex mechanism, possibly based on message passing, handshaking and FIFOs, while it is a simple data flow in figure 6 synchronized by the appearance of values. New functions can easily be added independent of what exists already. E.g. if we want to add a new performance monitor function `pm` in parallel to the `oam` function,

```
nt (i1, i2, mi) = (o1, o2, mo)
where   (o2, s4, s10) = oam (i2, s3, s9)
        (s3, s1, s5) = switch (s2, s4, s6)
        (o1, s2, s7) = oam (i1, s1, s8)
        (s11, s8, s5, s9) = management (mi, s7, s5, s10)
        s12 = pm (i1)
        mo = merge (s11, s12)
```

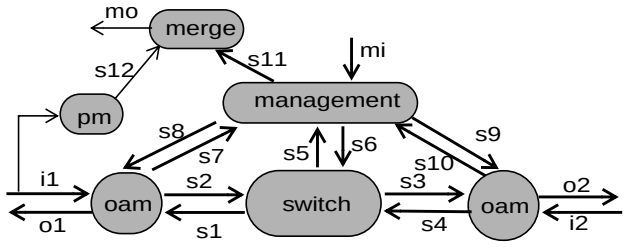

Figure 7. Functional specification with performance monitoring function

which operates on stream `i1` and contributes to stream `mo` to inform the external management system, it is straight forward as shown in figure 7. `merge` is a function which merges two streams. In fact, functions, which operate on any of the streams can be added or removed without complication. Some care has to be taken when merging and splitting streams, but since these streams are essentially very simple data types, resembling a simple communication mechanism, it typically does not take more than a few seconds to accomplish even complicated merge operations.

This adding and removing of functions can be done at any hierarchical level, e.g. the `pm` function could be inserted inside the `oam` function at the next lower hierarchical level.

Similarly, sub-functions can be moved from one function to another, if the involved data dependences allow this. E.g. a sub-function of oam could be moved into the switch function, if it only operates on the streams connecting oam and switch. Such a transformation could modify the interfaces of functions, but in a predictable and automatable way. Essentially, the ease of these transformations comes from the fact, that the communication mechanism inside a function between sub-functions is identical to the mechanism between the functions themselves. Thus, the move of one sub-function from one block to another is merely a re-grouping of the functions but does not change the data and communication flow. A functional description can be viewed as a tree of functions as illustrated in figure 8. A partitioning

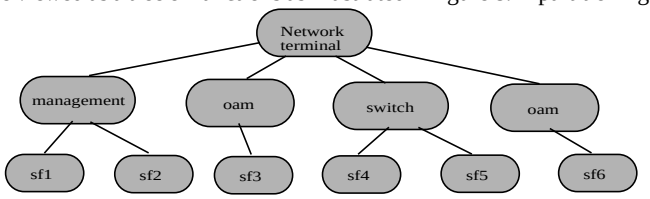

Figure 8. Function call tree

into blocks corresponds to a cut line in the function call tree but does not influence the data flow between functions. All functions are side effect free, thus a re-grouping or re-scheduling does not influence the result as long as the data dependences are observed. Moreover, functions can be moved between hierarchy levels along the data flow arcs. For instance, the function `sf5` could perhaps be moved up to the `Network terminal` function, if the data dependences allow this. This would not change the computation but only where a value is computed. However, such re-grouping and function movement will influence the cost and performance in the implementation, because there communication inside a block, e.g. an ASIC, is of a different nature with different physical properties than across block boundaries. And this is also the case in process or object based models. If we take again SDL as example, we notice, that the communication inside processes is shared memory based and between processes it is based on asynchronous message passing.

To summarize, a functional description facilitates an efficient functional exploration essentially because of the simple and very abstract communication mechanism and because it does not imply an architecture of the final implementation.

The major problem and the reason, why such models are not in widespread use, is the distance to any possible implementation. As a major design decision an architecture must be proposed, which essentially means to develop an object based model as we have discussed it in the previous section. Only from such a model we can reason about the feasibility of the system and if performance, cost and other requirements can be met. We cannot do anything like that for the model in figure 5.

3. The Integration of Functions and Objects

Let's assume for a moment, that we have developed both, the functional and the object based model in order to gain the benefits from both. The next problem is what Selic et al. [15] described accurately as discontinuity. We have two models representing the same system, but we do not have a formal relationship between them. Thus, it is difficult to establish some sort of equivalence which makes validation difficult. But what is even worse, we cannot efficiently transform one model into the other, which essentially leads to the single-model-syndrome [15]. Any new modifications, which are unavoidable for large and complex systems, will only by done for the objects based model but not for the functional model. After a short time the functional model does not reflect the intentions and the state of development and must be abandoned. The maintenance of two separate system level models is too costly in today's hard pressed

product development projects.
We conclude from these observations and arguments, that even though it is desirable to develop a functional and an object based model, it is very costly and difficult to justify it in a project. The consequent decision is to develop only the model, which is essential for the design and implementation, which is the object based model. While this is understandable from a project leader's point of view, let's investigate potential benefits from a functional model, before we suggest a desirable methodology and identify urgent problems to be solved.

- A functional description is inherently simpler, thus faster to develop, because it implies fewer design decisions and, consequently, avoids all details resulting from these decisions. Most apparent is the absence of internal communication. Internal communication is a major issue when developing object based models and it is a major hurdle when the architecture should be modified.
- A design exploration in terms of functions is necessary in an early phase when the exact functionality of a product is determined. Fred Brooks noted 11 years ago [1]: "The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is so difficult as establishing the detailed technical requirements.... No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.". This functional exploration is supported by a functional description because it is relatively easy to add, modify, and remove functions. As argued by Lawson [11] unnecessary functions contribute significantly to unnecessary complexity with all its associated risks. A functional model makes the absence or presence of functions more obvious. It makes it also easier to add and remove functions.
- A functional description is of enormous value for validation and test of any architecture and implementation because testcases and test patterns for a system level functional validation can be derived directly from a functional system model. Functional system validation is a major challenge and the definition of a minimal set of testcases, which basically define when a system behaves correctly and when not, is a non-trivial task [10]. If a description of system functions does not exist, it must be developed and formulated as testcases.
- A functional system model allows to support the development and maintenance of product families. Very often a product is developed in different versions, each version addressing a different market segment, realizing different subsets of the full possible functionality, and fulfilling different requirements and trade-offs. Each version may require a different architecture but many of the functions will be identical. Hence, what makes a functional description useful for functional design exploration, facilitates also the derivation and validation of new versions.

At this point we face a dilemma, because we cannot exploit the benefits of a functional model unless we integrate the functional with the object based model. There are two ways out: One is to obtain the discussed benefits without a functional model in some other way, e.g. by developing efficient techniques to transform an object based model into another in order to explore the functional design space at that level. The other possibility is to derive alternative object based models from a functional model efficiently and essentially eliminating the discontinuity. Both alternatives are researched to some extend but face considerable difficulties.

3.1 Transformation between object based models

Transformations between object based models face the difficulty that certain design decisions are implicit present in an object based model, which cannot be undone easily. For instance, the splitting and merging of processes is difficult and inefficient and often results in solutions which have too high cost and low performance. Furthermore, the move of a process part from one process to another is often not a possibility at all, because processes are often modelled in a sequential way by means of a finite state machine or an algorithm. These models tend to have many control dependences which makes them hard to parallalize, as we know from research on parallalizing compilers [16].

3.2 Transformation from a functional to an object based model

Transformations from a functional to an object based model face the fundamental difficulty that they involves design decisions which inherently cannot be done by a method or tool, which means the discontinuity between functional and object based models has a fundamental nature. One way to cope with this is, to allow to formulate design decisions in a very compact way to capture the essence of the decision, and handle all the implications of the decision automatically. For instance, a design decision might be to use a handshake protocol with a 16 bit wide data bus as a communication protocol between two blocks. The consequences of this decision would be the implementation of such a protocol with all the necessary control to support the data flow between the two blocks that is required by the functional model. This would avoid a discontinuity in the design process because the object based model is automatically derived from the functional model, with the design decisions being an additional input. Moreover, it would facilitate efficient design space exploration because different architectures and communication schemes and their consequences in terms of performance and cost could be explored systematically and rapidly.
Another way would be to assume a certain target architecture and accept to generate inferior implementations in some cases, based on the hope that the selected architecture is well suited for most of the cases. Researchers in high level synthesis have selected this approach with some success.
Ideally, we would like to have full control over the transformation process without restricting the target architecture. Full control is only useful, if we have a reasonable method which defines which transformations should be applied, which requires that we also have a metric which allows us to "measure" a model with respect to certain implementation properties. Such a metric could guide the application of transformations, i.e. the transformation process. For this we need on one hand a set of transformations, which can be used to gradually transform a functional model into an object based model, which facilitates an efficient implementation. On the other hand we need a metric or estimation technique to analyse intermediate models. Some work in this direction has been done in the area of parallel programming [16, 18], where the challenge is similar to what we face. The objective there is to map a functional description of a program efficiently on different parallel architectures.

4. Hypothesis

We formulate our claim as hypothesis and describe an experiment that could support or falsify it. The hypothesis is: (1) With a functional model more alternatives with different functionality can be simulated than with an object based model. (2) With an object based model more accurate estimates about implementation properties such as cost, performance and power consumption can be achieved than with a functional model.
In the proposed experiment a requirements definition document is given to two separate groups of designers, group A should work with a functional model and a functional language such as Haskell or ML. Group B should work with an object based model and a language such as SDL, Erlang, or SpecChart. In a first phase both groups should develop an executable model with the desired functionality. The

desired functionality should be defined at the beginning of the experiment but not given to the designers. It should be gradually revealed as response to their questions and their developed models. It must be compatible with the requirements definition. This process is supposed to mirror the common situation, that a specification has to be developed based on a description of requirements and constraints which allows for many different interpretations. In a second phase of the experiment several but at least five different functionalities, which are variations of the first functionality, should be developed. In both phases the development time should be measured and compared for both groups of designers. Care must be taken when the designers are selected, to avoid any systematic bias towards one or the other methodology. Ideally a large number is chosen randomly. If this proves to be difficult, a second experiment with a comparable but different system and the same designers but in opposite roles can be conducted. The results must be evaluated with established statistical methods.

The second part of the hypothesis has the difficulty that we cannot foresee all the estimation techniques that are possible and may be developed in the future. So we assume that there is some kind of linear relation between the estimation error of experienced designers and the estimation error of the best possible estimation technique. Based on this we give a sample of models, say at least five functional models and five object based models, to at least five experienced designers, and ask them to estimate the size, performance and power consumption of possible implementations by giving a range, not only one specific figure. Then we ask other designers, again at least five for each model, to develop an implementation. Both, the estimate and the development should be with regard to design constraints and objective, e.g. to minimize cost while achieving a certain performance. In the experiment we measure the estimation ranges, the time it took the designers to derive the estimates, and how far the estimates are off from the implementations. All three figures should be significantly different for the functional and the object based models. The size of the models must not be too small so that the estimation is not a trivial task. Such an experiment might be relatively expensive. But if this is a relevant and important question, the costs of the experiment are more than worth it, and other scientific disciplines like psychology, medicine, pharmacy, physics, or biology, spend orders of magnitude more time and money on well planned experiments than computer science does. We argue that this is a very important question, which should be decided by experiments, because the investments in a methodology, languages, and education, which are based either on the assumption that our hypotheses is true or that it is wrong, are already enormous today and will be even larger in the near future. Furthermore, many research activities and directions assume one or the other answer of the hypotheses without verifying it by means of an experiment.

5. Conclusion

In essence we argue for a design methodology with two distinct specification models, a functional model for functional design space exploration, and an object based model for the implementation specification. We have described the benefits of each of these models which motivate the ambition to integrate them. The main challenge of integration is the fact, that an object based model contains many more design decisions than a functional model. The most promising way of integration is to represent design decisions in a compact way and independent from the consequences of these design decisions. This would allow designers, or smart decision making tools, to make a design decisions, perhaps in an interactive manner, and the consequences of these design decisions are implemented automatically by a tool.

Important steps to such an integration have been done already. Models of computation for a functional specification have been researched and good candidates have been identified [6, 14, 16]. Design transformations for various semantic models have been researched and formulated [13, 14, 16], and the formulation of specific transformations for the design of electronic systems is arguable a challenging but feasible task.

Finally, we formulate our claim as hypothesis and describe an experiment to justify or falsify this hypothesis. We believe, such an experiment, based on well established scientific principles, is essential to direct future research and industrial investment.

6. References

[1] Frederick P. Brooks, Jr., “No Silver Bullet”, *IEEE Computer*, pp. 10 - 19, April 1987.

[2] J.P. Calvez, *Embedded Real-Time Systems*, J. Wiley&Sons, 1993.

[3] M. Chiodo, P. Giusto, A. Jurecska, and M. Marelli, “Synthesis of Mixed Software-Hardware Implementations from CFSM Specifications”, *Proc. of the Int. Workshop on HW/SW Codesign*, 1993.

[4] T. DeMarco, *Structured Analysis and System Specification*, Yourdon Inc., New York, 1978.

[5] W. Ecker, “Using VHDL for HW/SW Co-Specification”, pp. 500-505, *European Design Automation Conference*, 1993.

[6] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, “Design of Embedded Systems: Formal Models, Validation and Synthesis”, *Proc. of the IEEE*, vol. 85, no. 3, 1997.

[7] D.D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems*, Prentice Hall, 1994.

[8] W. Horn, *Modelling of an ATM Multiplexer in a Network Terminal for a Mixed Hardware/Software Implementation*, Master thesis, Royal Institute of Technology, Stockholm, report no. TRITA-ESD-1998-06, May 1998.

[9] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard, *Object Oriented Software Engineering: A Use Case Driven Approach*, Addison Wesley, Reading, Massachusetts, 1992.

[10] A. Jantsch, J. Notbauer, and T. Albrecht, “Testcase Development for Large Telecom Systems”, *Proc. of the Int. High-level Design Validation and test Workshop*, November 1997.

[11] H. W. Lawson, “Salvation from System Complexity”, *IEEE Computer*, pp. 118 - 120, February 1998.

[12] A. Olsen, O Færgemand, B. Møller-Pedersen, R. Reed, J.R.W Smith, *Systems Engineering with SDL-92*, North Holland, 1995.

[13] A. Pettorossi, M. Proietti, “Rules and Strategies for Transforming Functional and Logic Programs”, *ACM Comp.Surveys*, June 1996.

[14] H. J. Reekie, *Real-time Signal Processing*, Ph.D. thesis, University of Technology at Sydney, 1995.

[15] B. Selic, G. Gullekson, and P.T. Ward, *Real-Time Object-Oriented Modelling*, John Wiley & Sons, 1994.

[16] David Skillicorn, *Foundations of Parallel Programming*, Cambridge University Press, 1994.

[17] A.C. Sodan, “Yin and Yang in Computer Science”, *Communications of the ACM*, vol. 41, no. 4, pp. 103 - 111, April 1998.

[18] M. Südholt, *The Transformational Derivation of Parallel Programs using Data-Distribution Algebras and Skeletons*, Ph.D. thesis, Fachbereich der Technischen Universität Berlin, 1997.

[19] K.Weidenhaupt, K.Pohl, M.Jarke, P.Haumer, “Scenarios in System Development:Current Practice”, *IEEE Software*, March 1998.

[20] Wayne Wolf, “Object oriented Co-synthesis of Distributed Embedded Systems”, A*CM Transactions on Design Automation of Electronic Systems*, vol. 1, no. 3, July 1996.

[21] P. Zave, “Classification of Research Efforts in Requirements Engineering”, *ACM Computing Surveys*, December 1997.