The Rugby Model: A Conceptual Frame for the Study of Modelling, Analysis and Synthesis Concepts of Electronic Systems

Axel Jantsch[1], Shashi Kumar[2], Ahmed Hemani[1]

[1] Royal Institute of Technology, Stockholm, Sweden
[2] Indian Institute of Technology, Delhi, India

Abstract

We propose a conceptual framework, called the Rugby Model, in which designs, design processes and design tools can be studied. It is an extension of the Y chart and adds two dimensions for design representation, namely Data and Time. The behavioural domain of Y chart is replaced by a more restricted domain called Computation. The structural and physical domains of Y chart are merged into a more general domain called Communication. A fifth dimension deals with design manipulations and transformations at three abstraction levels.
The model shall establish a common understanding of modelling and design process concepts for communication and education in the community. In a case study we illustrate how a design can be characterized with the concepts of the Rugby model.

1. Introduction

To categorise, conceptualise, and visualize problems in design automation and solutions to them we need a *conceptual framework*. Since 1983, when the Y chart was proposed as the conceptual framework for VLSI design, the complexity in terms of transistor count has increased more than two orders of magnitude, which raises new design issues that are not naturally modelled on the Y chart. For instance, modelling and refining communication and data elements into implementation primitives in a HW/SW codesign process is more complex than in the traditional HW design. Moreover, HW/SW codesign requires segregation of the design process at lower levels into separate HW and SW design flows while integration at higher levels of abstraction.

The Rugby model, presented in this paper, is a conceptual framework which (a) treats *design modelling* and *design process modelling* as two separate but interdependent issues, (b) covers and relates all design phases from requirements to implementation, and (c) allows studying the HW/SW codesign process.

The Rugby model adds two dimensions for design representation, namely *Data* and *Time* to the Y chart. The behavioural domain of the Y chart is replaced with a more restrictive *Computation domain*. The structural and physical domains of the Y chart are replaced by a more generic *Communication domain*. The importance given to the general concept of communication reflects the shift in focus of today's designers from layout and structure to system level communication aspects.

Furthermore, the model adds an orthogonal dimension to the design modelling plain to represent *design manipulations* at several abstraction levels. While the Y chart implicitly mixes design representation with design process aspects, the Rugby model separates these issues.

2. Existing Models

The Y chart [1] has three domains of design description: Behavioural, Structural and Physical. Each domain has many levels of abstraction. The design process is represented by step-wise refinement in all the three domains from outer levels towards the centre. It is possible to represent synthesis, design verification and analysis tools for VLSI design on the Y chart. However, the Y chart does not have any explicit representation for time, communication and data abstractions. Modelling HW-SW systems and codesign problems is also not possible.
X- Chart [3] is an extension of the Y chart, and includes an additional aspect of Testing but lacks means to model timing, timing constraints and data abstractions.
Multi-Level Cybernetic proposed by Ramming [4] can model the design processes and design strategies, but lacks features for modelling designs. A design process is modelled as a composition of level invariant activities like optimization and design modification, and level variant activities like synthesis. The main emphasis is on the complete design process and on coarse grained process steps, while our main focus is on design representation and on

fine grained design manipulations.

Design Cube [2] is a framework to model design activities in the VHDL environment and unlike other frameworks also has the capability to model time and data at different levels of abstraction. However, it lacks the ability to represent the physical implementations.

While explicitly representing Time and Data in the Design Cube represents progress, both the Y-Chart and the Design Cube mix design modelling with design process. The current models are weak in describing mixed HW/SW designs and HW/SW codesign processes. While the Y chart and the Design Cube reflect the hardware design issues, our model is able to represent HW/SW systems and design process by virtue of having explicit data and communication domain.

Furthermore, by introducing a new axis of design transformations we separate the issues of design modelling and design process modelling. We claim that design manipulations and transformations can also be investigated at several abstraction levels.

3. The Rugby Model

Fig. 1 gives a graphical overview of the proposed model.

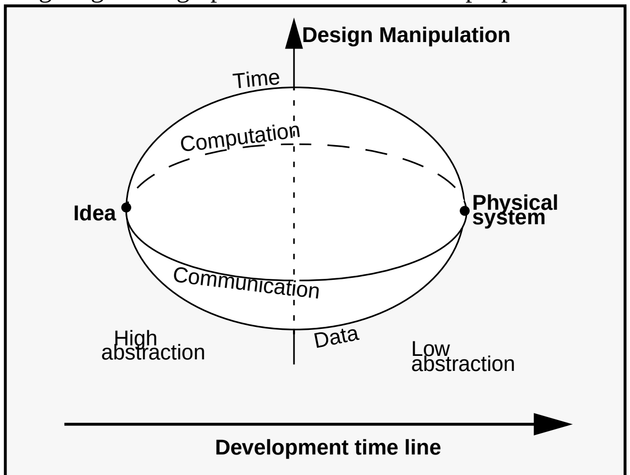

Figure 1. The Rugby model

The development of a new design/product starts with an idea. The idea generally represents informal functional and non-functional requirements. This idea goes through many refinements before a final physical product comes into existence. In our model, the design refinements are carried out in four domains, namely computation, communication, data and time. At the start of the design process these four aspects of the design are not separable. We represent the idea stage of the design as a *point* in our model.

As we go further in the design process, different aspects of the design (or its model) are more clearly distinguishable. Design functionality is partitioned into sub-functions/sub-components/processes which may need to communicate with each other to achieve the functionality. The evaluation of functions require computation on data. The design also may have real time requirements on the evaluation of functions. As we further refine the design towards physical implementation making decisions like hardware/software implementation of the function, or implementing functions using a library of logic blocks, the distinction between these four aspects become wider and clearer. Finally, when the design takes the shape of a physical silicon chip with various layers of dopings and metallization, the distinction between these domains again disappear. Therefore, we represent the design process as four lines initially diverging from a point (representing idea) and later converging to another point (representing the final product). In each domain, the point representing the idea is at the highest level of abstraction, whereas, the point representing the physical system is at the lowest level of abstraction.

A design process consists of a number of **design manipulations**, which refine "the idea" to the final product. In our model we also recognize various abstraction levels in design manipulations and represent it as a separate axis which is truly orthogonal to the modelling axes. At the lowest level we have design manipulations which deal with specific design *instances*, like design entry editors; at the middle level we put design manipulation algorithms like high level synthesis *algorithms* or test pattern generation algorithms; at the highest level of abstraction in the design manipulation abstraction we have *methodologies* which guide the design process from idea to physical product.[1]

3.1 Design Representations

To model different aspects of mixed HW/SW systems and analyse their different problems, we choose Computation, Communication, Time, and Data as the four domains in our model.

Figure 2 magnifies part of the domain lines of the Rugby model and shows the abstraction levels from abstract requirements definitions to a concrete mixed HW/SW implementation. It illustrates that domain lines can split when design activities specialise. However, for each split there must be a corresponding joining of lines during system integration, which is not shown in figure 2.

Due to lack of space we cannot give a full account of all the abstraction levels depicted in the figure, but we only discuss a few interesting aspects.

1. It has also been suggested to use the name "Universe model" with the "idea" corresponding to the big bang and the "physical system" to the big crunch. The modelling lines would correspond to different physical forces and the "design manipulation" axis to mathematical and physical laws.

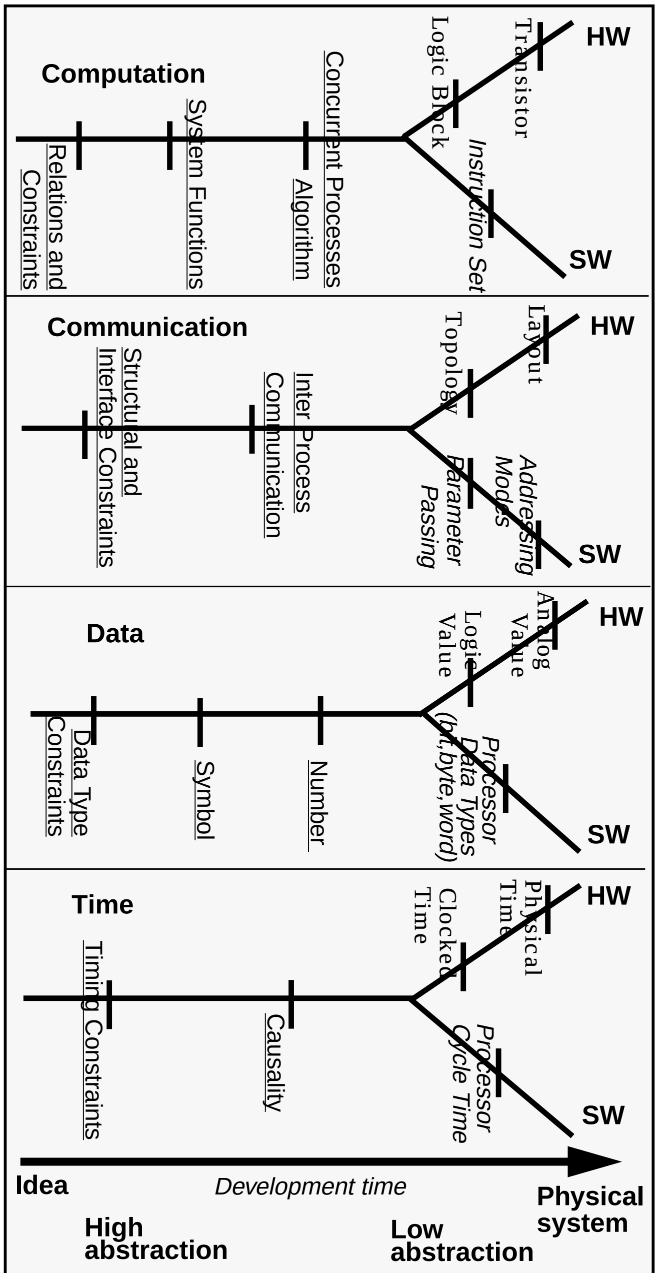

Figure 2. Abstraction levels in modelling domains

3.1.1 Computation. The computation domain is derived from Y chart's behavioural domain but is more restrictive and focuses on the way the results are computed independent from the exact data types involved and from the exact timing behaviour of the computation. It is concerned with the relationship of input and output values, i.e. the behaviour as it is observable from the outside.

The instruction set level is the lowest abstraction for software. Though some computational concepts like sequencing, branching and sub-routines are similar to algorithmic level but is considered less abstract than algorithmic level because (a) the control elements are more primitive and (b) it is processor specific.

At the system function level the system is described from a purely external view without considerations of the partitioning and the implementation of the system. The difference between the "system functions" and the "relation and constraints" level is analogous to the difference between a function and a relation in mathematics. A function maps a given input to one specific result. A relation maps an input to a set of results, i.e. a relation allows many more solutions.

3.1.2 Communication. Communication is independent from computation, data, and time. Complex systems are naturally modelled as communicating concurrent processes. Refining these abstract communications to intra and inter component (ASICs, processor cores, memories etc.) communication primitives is a major part of the design effort and is now being treated as a research problem [7, 8]. Furthermore, languages and notations that were not main-stream in the hardware design community, are being explored to specify communication dominated functionality [9, 10]

The structural and physical domains of the Y chart are merged into this domain because movement from a topological to a layout model is considered as a refinement operation, not an inter-domain movement. The layout level is based on principles of geometry and uses physical units to describe geometric parameters. This level corresponds to the physical domain in the Y chart. The topological level is only concerned with the presence or absence of connections between design elements. It corresponds to the structural domain in the Y chart. The inter process communication level is concerned with mechanisms and protocols of communication between design elements. At the highest level only the interface and communication constraints are expressed, to which every implementation of the interfaces must comply.

3.1.3 Data. For software there is a long tradition to model data types explicitly with modelling concepts such as entity-relationship diagrams, and in the hardware community data structures have evolved from bit vectors to more complex data structures like arrays, records, linked lists etc. This is reflected in a flurry of research activity in this domain, e.g. [11, 12]. Thus, for the modelling of mixed HW/SW systems it is desirable to treat data and data types as an independent aspect.

3.1.4 Time. Time is a crucial design characteristic which deserves independent analysis. Many electronic systems are expected to be reactive real-time systems with soft or hard real time constraints. Furthermore, numerous publications [5, 6] on how to model time illustrate that it is not bound to a particular kind of computation, but rather it is independent. In the formulation of the abstraction levels we follow mostly [2] but extend it to software on one hand and include timing constraints on the other hand.

3.2 Abstractions of Design Manipulations

Orthogonal to the modelling domains is the design manip-

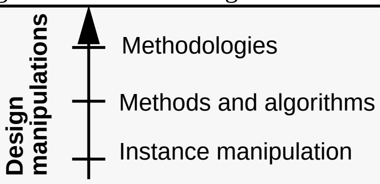

Figure 3. Abstractions of design manipulations

ulation, which can also be represented at several abstraction levels. At the lowest level, design entities are directly created, transformed, and removed using schematic and text editors. At the next higher level, algorithms and methods are employed as in place and route, logic and high level synthesis. At the highest level, the algorithms and methods from the second level are used and combined into methodologies which cover the entire design process.

All the four design modelling domains are relevant at each level of the design manipulation axis. For example, instance manipulation editors can operate on stick diagrams as well as graphical system description diagrams in SDL. However, at higher levels of the design manipulation axis several design models and abstraction levels may simultaneously be used.

Today's level of automation and tool sophistication decreases with higher levels. We have sophisticated graphical and textual editors for instance manipulations at all abstraction levels for design models, we have synthesis algorithms addressing some but by no means all important transformation problems, but we hardly have any tool support to describe and execute methodologies. The level of tool support is also different for different abstraction levels of the design modelling domains, with typically higher degree of automation for lower levels. This is indicated in figure 4.

Today there is no systematic and well understood relation between the three levels Instance Manipulation, Synthesis Algorithms, and Methodology. We have no general method to construct a synthesis algorithms from instance manipulations or to deploy synthesis algorithms to compose an executable methodology. We have point solutions but no general understanding and concept of this task. Figure 4 shows how design automation has evolved over the last few decades.

3.3 Unexplored Problems and Issues

The model can be used to represent many established methods and tools. In addition, many open problems are identified. We only sketch a few of them.

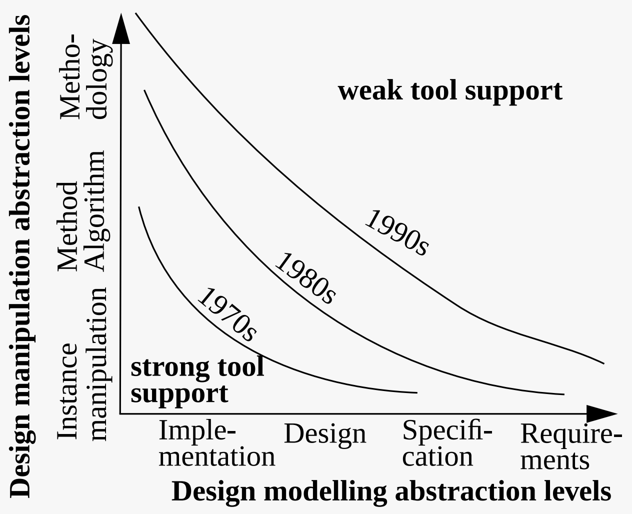

Figure 4. Automation and abstraction levels

- **Data type refinement**: At lower levels of the Data domain the refinement of data is handled by state encoding and technology mapping techniques. But higher levels models, which typically use abstract symbols for performance modelling, based on queueing theory or petri nets, have usually no direct link to models with more concrete data types to be used in the design and implementation process.
- **Transformation of system functions into concurrent objects**: In early design phases a system is best described in terms of use cases [13], scenarios [14], and system functions. These functions are independent from each other and represent the requirements. The challenge to transform these system functions into interconnected concurrent objects is a key to link a problem oriented description to an implementation oriented model. It has not been systematically addressed.
- The task of **transforming requirements definitions** into a system which fulfills the requirements is notoriously difficult. The Rugby model indicates that we need to address this problem in all four domains computation, time, data, and communication. Computational constraints in terms of input/output relations and timing constraints have to some extent been researched. But we have neither a formalism to express data type constraints and structural and interface constraints, nor methods to integrate these constraints into the design process.

4. Case study

The design of a network terminal (NT) [15] unit serves as a case study. The NT provides the interface between the public access network and the private customer premises network (CPN). Physically it is a device that is installed at the customer's premises and it allows a user to access the distribution network with its variety of services. The NT provides an inter-

Table 1. Models in the Network Terminal development

Abstraction level			Design modelling domains: Computation	Communication	Data	Time
	Requirements		functional constraints	interface constraints	ATM cell definition	performance constraints
	System model		algorithm/FSM	inter-process communication	symbols	causality
	SW	C model	algorithm	parameter passing	symbols/numbers	causality
	SW	Assembler	instruction set	parameter passing	processor data types	processor cycle time
	HW	VHDL model	algorithm/FSM	topology	symbols/bits	clocked
	HW	Synthesized Netlist	logic blocks	topology	bits	physical time

face to the access network and several interfaces to different CPNs, e.g. to an ethernet or to a telephone. The ATM Multiplexer translates the connection identifier fields in the headers of the ATM cells in order to deliver the cells to their correct destination.

Several models have been used or developed during this project, each of which is characterized in the following sections using the Rugby model. It has to be emphasized that the NT has been developed without awareness of the Rugby model. With respect to the kind of models developed and tools used, it was a very typical project. Also, the Rugby model was not influenced by this particular project. Nevertheless the NT models can be very naturally characterized and analysed by means of the Rugby model.

Table 1 summarizes several NT models from the requirements definitions to implementation models. Figure 5 illustrates these models in the Rugby diagram, but it does not distinguish between SW and HW models for the sake of clarity.

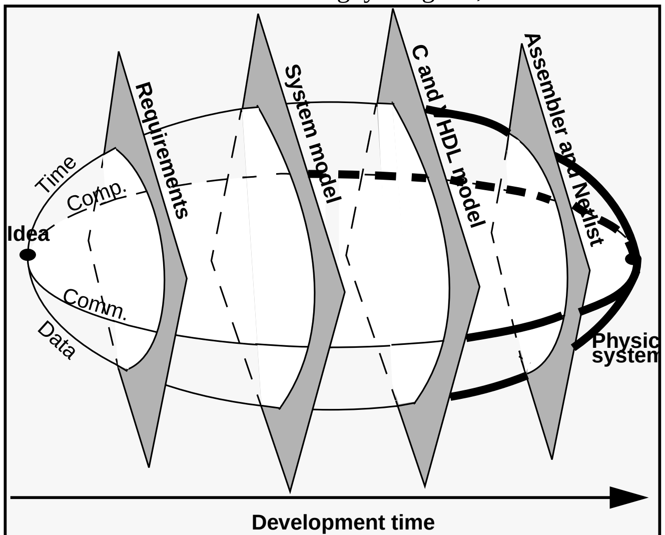

Figure 5. The NT models in the Rugby frame.

4.1 Requirement definitions

The requirements definitions were the input to the development activities. They consist of several documents, for instance ITU documents, books and article on ATM and communication networks and protocols. A significant part of it is informal based on existing experience and discussions with Ericsson engineers. From all these sources an understanding of the NT was developed within the group which can be viewed as the requirements definition even though there is no single explicit document. These requirements include functional constraints, interface constraints, data constraints, and performance constraints.

4.2 System Model

From the initial requirements we developed an executable model, the abstraction level of which was determined by the choice of the language, SDL. SDL is based on concurrent processes which communicate with each other via an asynchronous message passing mechanism.

The most important design decision during development of the system level model was the partitioning into concurrent processes. It also gives the frame for the HW/SW partitioning, which is performed at the process level granularity.

4.3 Software

4.3.1 C Model. The abstraction level in the computation domain is the algorithmic level. Without a multi-tasking OS all concurrent processes of the SDL model are flattened out into sequential C code. Hence, the abstraction level in the communication domain would be parameter passing between functions.

4.3.2 Assembler model. The compilation of the C code into assembler is typically a fully automated process which makes it very predictable. The computational elements are the instructions of the target processor which also determine the time abstraction and the data types. The communication abstraction is parameter passing.

4.4 Hardware

4.4.1 VHDL Model. The developed VHDL model is synthesizable with Synopsys' design compiler. Thus, it is RTL code with a clocked time and algorithms describing the computations inside the processes. The algorithms are written in a style that are interpreted by the design compiler as finite state machines. Thus, the description of the processes is very similar to the SDL models in terms of computation, control flow and data flow. The principal difference comes from the different level of abstraction in the time domain, causality in the SDL model and clocked time in the VHDL model.

The second important difference concerns the inter-processes communication, which is significantly refined in the VHDL model. In the VHDL model the topology is determined, describing precisely which port of one process connects to a port in another process. The asynchronous message passing mechanism in SDL is refined into handshake or finite FIFO based protocols.

4.4.2 Synthesized Netlist. The result of synthesizing the VHDL model with Synopsys' design compiler is a technology mapped netlist. The computation elements are logic blocks, the communication abstraction is a topology between the logic blocks as defined by the netlist. Note, that for the VHDL model the communication abstraction is also topology characterizing the connection between processes. In the synthesized netlist this topological model has been extended inside the processes leading to a topological hierarchy. All symbols have found a concrete bitvector representation and the timing has been refined into physical time units based on the delay of the elementary blocks.

In the next step of placement and routing the topology would be refined into a geometric model with physical units for the elementary cells and the wires, and the physical timing model would be extended to the interconnects.

4.5 Design manipulation

The design process can be characterized in terms of the abstraction levels of manipulations applied. In figure 6 is indicated when and in which domain an algorithmic method has been applied as opposed to a manual instance manipulation. For the lower levels, from the C and VHDL levels downward, automatic methods and tools were used. The transformation from the requirements definition to the system model was a purely manual activity. But the transformation from the system level to the separate HW and SW models was different for the different domains. In the communication and data domain

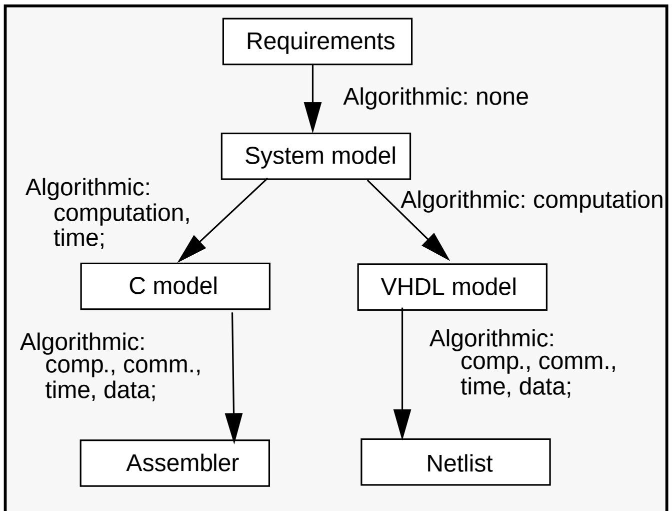
Requirements
Algorithmic: none
System model
Algorithmic: computation, time;
Algorithmic: computation
C model
VHDL model
Algorithmic: comp., comm., time, data;
Algorithmic: comp., comm., time, data;
Assembler
Netlist

Figure 6. Design manipulations in the NT development.

it was manual due to many design decisions taken. In the computation domain it was a systematic and mechanical transformation, although it was not done by a tool. In the time domain it was also a mechanic transformation for the SW model, whereas it was a manual step with significant design decisions for the HW part. This is also illustrated in figure 5 with bold lines indicating algorithmic transformation and thin lines indicating manual activities.

Figure 6 sketches also the development methodology by showing the design models and the kind of transformations applied on them. However, as in many other similar projects the methodology was derived in a simple way. From a template based on explicit guidelines and experience a project specific methodology was formulated intuitively and executed by the design team. It would be desirable to define general and versatile methodologies in a more formal way and be able to apply them classes of development projects. The execution of methodologies could also be formalized and automated to a higher degree. This would relieve the designers from the task of keeping track of numerous files and models created and used by various tools. They could rather focus on the design models and their more abstract and relevant properties.

Many point solutions address these issues but a common formal notion of a methodology and its execution encompassing multiple tools and vendors is still missing.

4.6 Discussion

The models in this case study can be characterized very naturally with the domains and abstraction levels of the Rugby model. Table 1 gives a concise and comprehensive view of different models and what kind of information is present. To convey the same amount of information, that is apparent in the table with the Rugby model as background, one would need lengthy explanations. This illustrates how the

Rugby model increases communication and understanding.

In addition to the information about the individual models, table 1 gives significant insight into the methods used for refinement one model into the next. For instance, the main difference between the SDL and VHDL model is in the communication and time domains, where the main effort in this refinement step has been spent. In the computation domain the mapping from SDL state machines to VHDL state machines was straight forward without involving any design decisions. Note, that this information is not inherent in the choice of the languages involved. If we had used Synopsys' behavioural compiler instead of design compiler to synthesize the VHDL code, the VHDL model would be at a different abstraction level in the time domain, because scheduling is a major task of the behavioural compiler, i.e. to transform a design from a causality level to a clocked time level in the time domain.

Another example is the refinement of the communication domain in the software part, where table 1 reveals where concurrent processes are merged into a sequential stream.

5. Conclusions

Conceptual frameworks need to evolve with the increasing complexity of electronic systems to properly model, represent and analyse designs and design processes. A unified framework for modelling of designs and design processes of today's digital electronic systems has been presented. It covers models of electronic systems from requirements specification to the implementation. Our treatment of Data and Time as independent domains makes it distinctively more powerful in modelling software systems, mixed HW/SW systems and HW/SW codesign processes. We also observe that at higher levels of abstraction, the modelling concepts are common for hardware and software systems. By having an independent domain for design manipulation, we can characterize the design methods and tools according to their manipulation power.

We believe a good conceptual framework makes the analysis and communication more efficient and a standardization of such a framework including terminology, on the pattern of the 7 layer ISO OSI reference model, would foster research and education.

6. References

[1] D. D. Gajski and R. H. Kuhn, "Guest Editor's Introduction: New VLSI Tools", IEEE Computer, Dec. 1983.

[2] W.Ecker, M.Hofmeister, and S.März-Rössel, "The Design Cube: A Model for VHDL Design Flow Representation and its Application", in *High Level System Modeling: Specification and Design Methodologies*, chapter 3, ed. R. Waxman and J.-M. Berge, *Current Issues in Electronic Modeling*, vol. 4, Kluwer Academic Publishers, 1996.

[3] F.J. Ramming, *Systematischer Entwurf Digitaler Systeme*, B.G. Teubner, Stuttgart 1989.

[4] F. J. Ramming, "A MultiLevel Cybernetic Model of the Design Process", Proc. of IFIP Working Conference on Methodologies for Computer System Design, 1985.

[5] B. Dasarathy, "Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them", *IEEE Trans. on Software Engineering*, Jan. 1985.

[6] J.F.Allen, "Towards a General Theory of Action and Time", *Artificial Intelligence*, vol. 23, 1984.

[7] J.A. Rowson and A. Sangiovanni-Vincentelli, "Interface-Based Design", *Design Automation Conference*, 1997.

[8] J.-M. Daveau, G.F. Marchioro, T. Ben-Ismail, and A.A. Jerraya, "Protocol Selection and Interface Generation for HW-SW Codesign", *IEEE Transactions on VSLI Systems*, vol. 5, no. 1, pp. 136 - 144, March 1997.

[9] A. Seawright, U. Holtmann, W. Meyer, B. Pangrle, R. Verbrugghe, and J. Buck, "A System for Compiling and Debugging Structured Data Processing Controllers", *Proceedings of Euro-DAC 96*, September 1996.

[10]J. Öberg, A. Kumar, and A. Hemani, "Grammar-based Hardware Synthesis of Data Communication Protocols", *Proc. of Int. Symposium on System Synthesis*, 1996.

[11]G.deJong, B.Lin, C.Verdonck, S.Wuytack, and F.Catthoor, "Background memory management for dynamic data structure intensive processing systems", in *Proc. of Int. Conf. on Computer-Aided Design*, November 1995.

[12]F.Franssen, F.Balasa, M.van Swaaij, F.Catthoor, and H.De Man, "Modeling Multi-dimensional Data and Control Flow", *IEEE Transactions on VLSI Systems*, 1(3):319--327, 1993.

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

[14]C. Potts, K. Takahashi, and A.I. Anton, "Inquiry-Based Requirements Engineering", *IEEE Software*, March 1994.

[15]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.