# Rugby: A Meta-Model to Study Concepts in Electronic System Design

Axel Jantsch, Shashi Kumar, Ahmed Hemani

***Abstract**: We propose a conceptual framework, called the Rugby Model, in which designs, design processes and design tools can be studied. The model has similar objectives as the well known Y chart [1] but its scope is extended to handle designs and design processes required for complex systems requiring concurrent processes and mixed HW/SW implementation. The Rugby model has four domains, namely, Computation, Communication, Data and Time. The behavioural domain of the Y chart is replaced with more restricted computation domain. The structural and physical domain of the Y chart are merged into a more generic domain called Communication. The new domains Data and Time have become necessary to model data abstractions used at various levels of design, and to explicitly model timing constraints at various levels in the design process, respectively. We show that the Rugby model is able to represent mixed HW/SW designs and design processes for HW/SW codesign at various levels of abstraction. It not only can represent state-of-the-art of current electronic systems and electronic system design automation, but it also points to gaps in the availability of tools and methodologies for designing complex system.*

## 1: Introduction

A conceptual framework like the Y chart [1] helps designers, researchers, teachers, tool developers, and foremost students, to conceptualise, categorise, and visualize issues in design automation. Since the introduction of Y chart in 1983, 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. Today, the era of system on a chip (SoC), with many concurrent and communicating activities on a single device, has emerged. The consequence is that we need to revise the meaning of the old domains like “behavioural”, “structural”, and “physical”. Also some new aspects like data, time and communication have become important when dealing with complex systems. While Y chart suffices to describe the process of HW implementation of an algorithm on a chip, a new model is required to deal with many communicating algorithms on a chip. A further complication is a heterogeneous implementation technology, which includes embedded software, DSP cores, micro processor cores, and custom hardware. For instance, HW/SW codesign requires segregation of the design process at lower levels into separate HW and SW design flows, and integration at higher levels of abstraction.

We must understand the three important concepts, namely, hierarchy, abstraction and domains before we present our model.

**Hierarchy:** *A hierarchy is a, possibly recursive, partitioning of a design model such, that the details of each part is hidden into a lower hierarchical level.*

Hierarchy defines the amount of information presented and visible at a particular hierarchical level of a model. At all hierarchy levels the same modelling concepts are used. The motivation for hierarchy is to hide information when it is not needed and to display details when they are useful.

**Abstraction:** *An abstraction level defines the modelling concepts and their semantics for representing a system. The type of information available at different levels is different. A higher level ignores some irrelevant information at a lower level and encodes it using different concepts.*

Abstraction defines the type of information present in a model. Unlike hierarchy, abstraction is not concerned with the amount of information visible, but with the semantic principles of a model. In general, the movement from high to low abstraction levels includes a decision making process. By making design decisions and increasing information about implementation details we replace more abstract models with less abstract models, until the system is manufacturable.

**Domain:** *A domain is an aspect of a model which can logically be analysed independently from other aspects.*

A domain focuses on one design aspect. Real models always contain several aspects or domains but different models may emphasize one domain more than another. Models, which focus on one particular domain use modelling notations and constructs to model the design aspect of concern explicitly. Other design aspects may be implicitly part of the models. Whereas hierarchy and abstraction simplifies the design, domain partitioning helps the developers of tools and methodologies to cope with the complexity. The domains considered in this article are computation, communication, data, and time.

While hierarchical partitioning is mostly a manual endeavour, the definition of abstraction levels and transformations between them, is behind most of the advances in design automation. While hierarchy is a general and important concept, it is not explicit in the Rugby model. We assume, hierarchy is possible at all abstraction levels in any domain.

Rugby covers and relates models of all design phases from requirements to implementation. Consequently, it also allows to study the HW/SW codesign process. For the sake of conciseness and brevity we restrict the discussion here mostly to issues at the core of design of digital hardware and software of embedded systems and systems on a single chip. The model derives its name from the similarity of its visual representation (see figure 1) to the shape of a Rugby, with the domain lines forming the seams. The paper is organized as follows: The next section presents other meta-models and discusses briefly their limitations. Section 3 describes the Rugby model with its four domains. Section 4 outlines the possible use by mapping design activities into the model and by identifying research topics suggested by the model.

## 2: Existing Models

Previously proposed meta-models have several limitations which we try to overcome in Rugby. Earlier models like the Y chart [1] did not pay attention to time, data, and communication aspects. Gradually, time and data abstractions have been introduced [2], but not consistently to cover all levels of representations from requirements to implementation. Communication has emerged as a hot research area, but meta-models have not yet appreciated the fact, that topological and geometric structures are only low level manifestations of the more general concept of communication.

The Y chart and the design cube do not separate design modelling issues from design process issues and can only be fully understood in the context of concrete methodologies and tools. Essentially, the domains and abstraction levels are chosen in a way to make them compatible with design languages, tools, and methodologies under consideration. While this has the direct benefit of providing a frame to categorize models, tools and methodologies at the same time, we believe that much can be gained by cleanly separating these issues. Thus, the Rugby model, as presented in this article, focuses solely on design modelling issues and selects the domains and abstraction levels only based on inherent properties of the models. In a variant of Rugby [3] we have introduced a fifth domain addressing the design process and we argue, that the design process domain is orthogonal to the design modelling domains and can also be viewed at several abstraction levels. However, for the sake of clarity and brevity we restrict the discussion in this article to the four modelling domains.

VSI's Model Taxonomy [10] has been developed concurrently with and independently from Rugby. It has similarities in the basic understanding of the important issues in electronic designs at system level. In particular, it also recognises the importance of the Time and Data domains and assigns them independent axes. But its objectives are focused on reuse of components and designs, whereas the Rugby model has the objectives to derive a better understanding of electronic design concepts and processes. This is also revealed by the different names used, e.g. precision level in the model taxonomy and abstraction level in Rugby. It is underscored by the determined attempt in Rugby to use clearly different concepts for different abstraction levels, while in the model taxonomy sometimes the same concepts with varying accuracy are used for several different levels, see for instance the temporal precision.

# 3: The Rugby Model

Y chart is biased towards hardware implementation because its three domains, behavioural, structural and

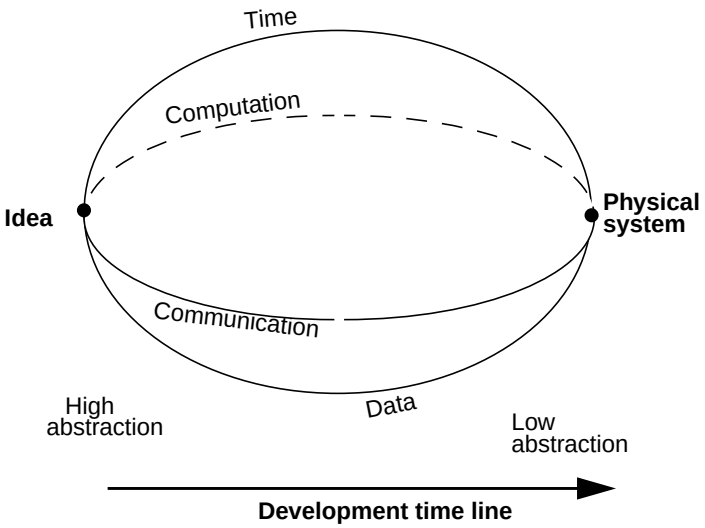


**Figure 1. The Rugby model**

physical, are inadequate to represent concepts such as inter-process communication, timing behaviour of systems, and various types of data abstractions and data encoding. To model different aspects of concurrent processes and mixed HW/SW systems and analyse their different problems, we choose Computation, Communication, Time, and Data as the four domains in our model (see figure 1).

In order to represent different implementation technologies such as hardware and software, we allow domain lines to be split. Hardware and software, when close to realization, use distinct concepts. For instance an assembler program and a netlist of gates are very different. On the other hand, at higher levels hardware and software developers use similar concepts such as communicating processes and abstract data types and can therefore be treated uniformly. Thus, at higher levels from requirements definition to communicating processes there is only one line in each domain representing the system level. This is independent from the later implementation which can be hardware, software or a mixture of both. At lower levels the characteristics of the implementation technology becomes dominant which requires separate domain lines as illustrated in figure 2.

## 3.1: Justification of Domains

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 of the exact data types involved and from the exact timing behaviour of the computation.

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. Thus, for the modelling of mixed HW/SW systems it is desirable to treat data and data types as an independent aspect.

Time is a crucial design characteristic which deserves independent analysis. Many electronic systems are expected to be reactive real-time systems or have hard real time constraints. Furthermore, numerous publications on how to model time illustrate that it is not bound to particular kinds of computation, but it is rather independent.

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 [5]. Furthermore, languages and notations that were not main-stream in the hardware design community, are being explored to specify communication dominated functionality [6, 7].

Figure 2 magnifies part of the domain lines of the Rugby model and shows the abstraction levels from

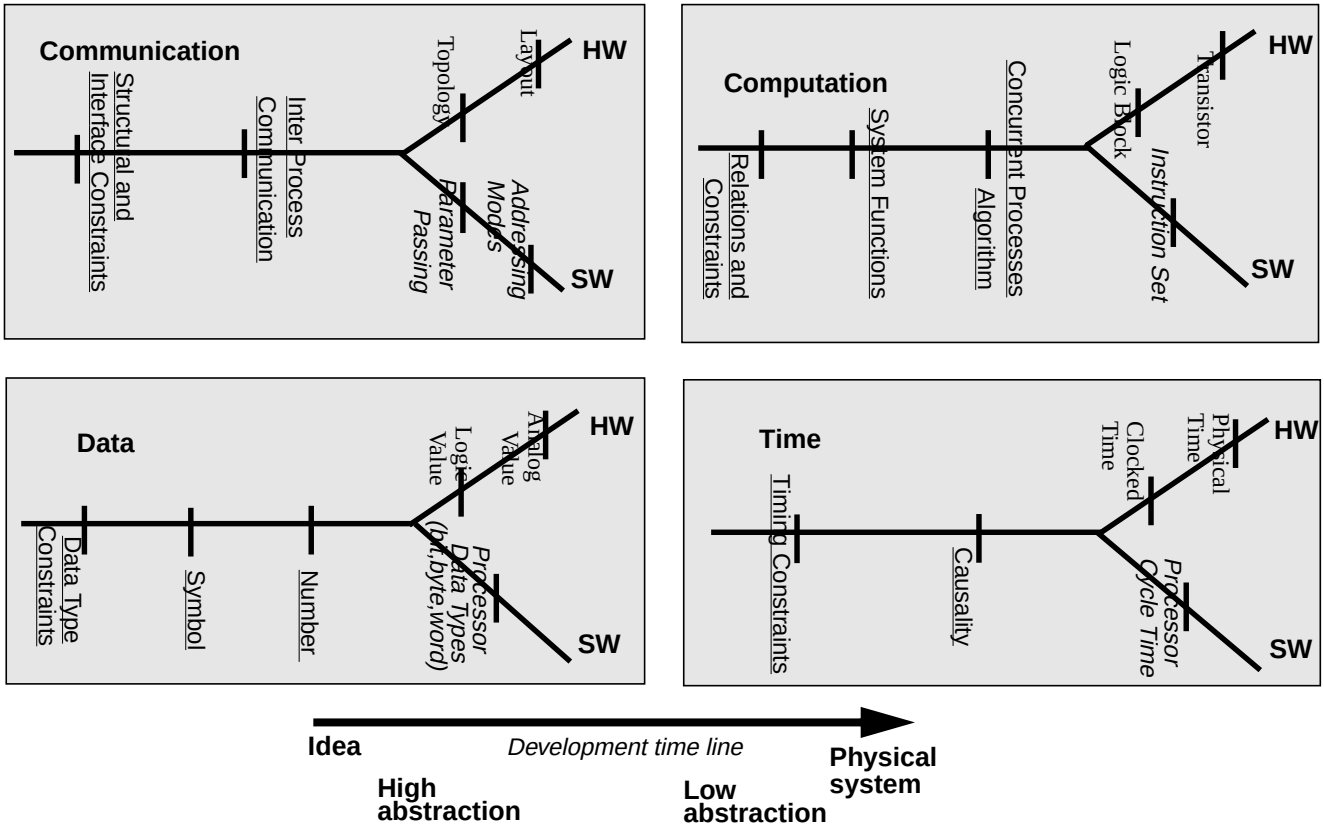


**Figure 2. Abstraction levels in modelling domains**

abstract requirements definitions to a concrete mixed HW/SW implementation. It illustrates that domain lines can split when design activities specialise. However, each split must have a corresponding joining of lines during system integration, which is not shown in figure 2.

## 3.2: Computation

The computation domain is concerned with the relationship of input and output values, i.e. the behaviour as it is observable from the outside. At the transistor level, models are based on differential equations representing

V-I characteristics. At the logic block level models are based on boolean functions.

The instruction set level is the lowest abstraction for software. Even though some computational concepts like sequencing, branching and sub-routines are similar at the algorithmic level, it is considered less abstract than the algorithmic level because the control elements are typically more primitive and it provides an abstraction layer around the details of the underlying processor architecture.

At the algorithmic level, models are based on control primitives such as ‘sequence’, ‘parallel’, ‘if’ and ‘loop’, and operators which manipulate data objects. Parallelism at this level is expressed in terms of concurrent processes, where each process is typically described as an algorithm.

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.

### 3.3: Communication

The communication domain is concerned with the connections and interactions between design elements. For hardware, the structural and physical domains of the Y chart are merged in this domain because movement from a topological to a layout model is considered as a refinement operation, not as 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.

In software there is no equivalent concept to the topology and layout levels of hardware but there exist two other levels, which are denoted as “parameter passing” and “addressing modes”. Parameter passing is the communication concept between functions and procedures in a sequential algorithm. Addressing modes is a concept used in the definition of instruction sets. They describe the communication between the computational and the storage parts of a processor.

The inter process communication level is concerned with mechanisms and protocols of communication between design elements. The implementation of inter process communication channels and the physical structure of the design is irrelevant at this level.

At the highest level only the interface and communication constraints are expressed.

### 3.4: Data

The data domain is concerned with data types and data objects which are transformed by active design entities. The analog value level is based on real numbers and is used to quantify physical units like voltage, frequency, etc. The logic value level is based on mathematical logic and is used to represent boolean and logic expressions. The number and symbol levels are based on number theory and set theory, respectively. In software the lowest abstraction level is based on data types of the target processor, which are typically bits, bytes, and words of varying length.

### 3.5: Time

The time domain is concerned with the time relation between activities. The physical time level uses physical time units and is based on physical principles. Propagation delay, as it appears in simulation languages for digital systems, is a simplification of the physical time and could be viewed as separate abstraction level. At the clocked time level all activities are related to a clocking scheme and are based on concepts of digital time. While there is no analogous concept to physical time in software, the processor cycle time corresponds to the clocked time in hardware. At the causality level the total ordering of events is replaced by a partial ordering defined by generation and consumption of data and by explicit control dependences. At the highest level the time is expressed through performance constraints like data rate or frames per second.

In figure 2 we have emphasized that at high levels there is no distinction between hardware and software, as a system can be implemented either way or as a mix of both. However, at lower levels the abstractions are different and are therefore separated. At the same time, all software rests eventually on hardware. These two different views are both valid and correspond to two active HW/SW codesign research areas: horizontal and vertical codesign. In the *horizontal codesign* one part of the system is implemented in software on a standard processor, and the other part is implemented in custom hardware, e.g. an ASIC. In the *vertical codesign* the system's functionality is implemented entirely in software, but the underlying processor is also developed at the same time and both, software and the hardware engine are designed and optimised at the same time.

Note that a model almost never uses elements from only one domain. In fact, most real models exhibit properties of all four domains, and can be characterized as a four tuple to indicate the abstraction level in the four domains.

## 4: Representation of Design Models and Activities

To illustrate the Rugby model we give a few examples showing how existing modelling efforts and design methods and tools fit into the Rugby. More importantly, we point out some areas and issues which have not yet been investigated thoroughly but which may be interesting areas of research according to Rugby.

### 4.1: Design Modelling

Many dedicated modelling efforts are characterized by a specific abstraction level in one of the four modelling domains. A few examples may illustrate the point.

- The use of abstract symbols for data without relying on specific data values is an important feature of performance modelling in early system design phases.
- The distinguishing element between register-transfer models and behavioural models in HW descriptions is that the former assumes a clocked time and relates all activities to time slots, while the latter is placed on the causality level of the time domain.
- System level activities are conveniently described in terms of communicating processes and the inter-process communication mechanism is often a distinguishing feature of a particular modelling approach, which is independent of the computation concepts they use. For instance some modelling approaches and languages are based on asynchronous, buffered communication (SDL, Erlang, CSP) while others are based on synchronous communication (μC++, Lustre, SIGNAL, Esterel).

Specific abstraction levels in the four domains are related to each other and are typically dealt within specific design phases.

### 4.2: Design Transformations

In addition to conveniently representing traditional HW synthesis and SW compilation activities, the Rugby model can also represent activities in the now established field of HW/SW codesign as illustrated in figure 3.

- HW/SW allocation and binding takes a description in terms of concurrent processes and generates a topology.
- HW/SW partitioning creates a relation between a set of concurrent processes and a topology.
- HW/SW binding refines this relation and creates a mapping between the processes and the topology such, that each process is bound to only one topological element.
- HW/SW Interface Synthesis refines the Communication domain from the "Inter-process Communication" level to the "Topology" level.

### 4.3: Design Analysis and Estimation

Analysis, normally involves multiple domains. For example, a conventional logic simulation and timing

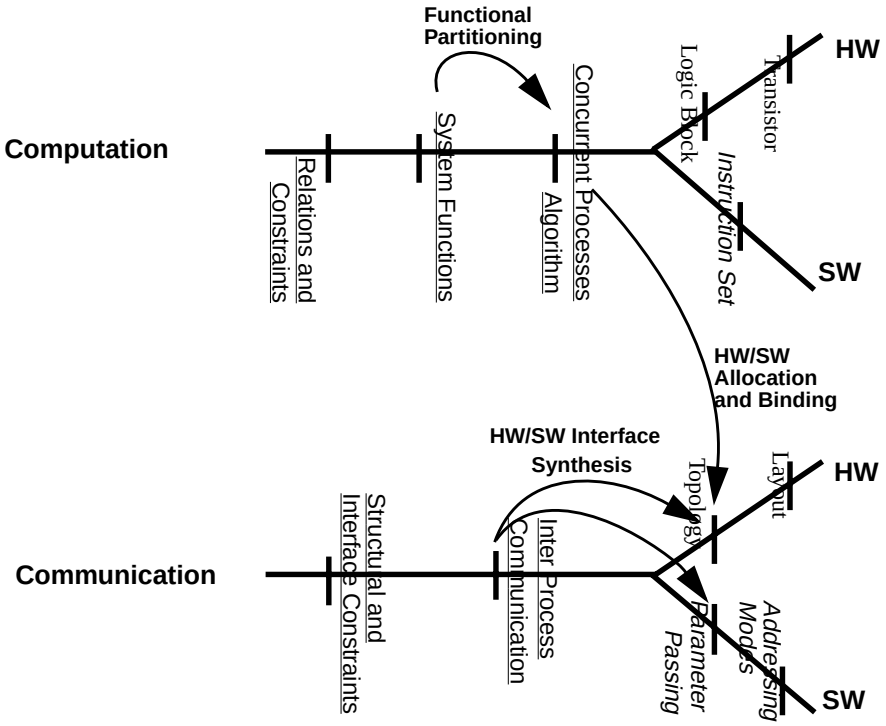


**Figure 3. HW/SW codesign tasks in the Rugby model**

analysis tool requires models at the logic block level in the Computation domain, at the physical time level in the Time domain, at the topology level in the Communication domain and at the logic value level in the Data domain as illustrated in figure 4.

At the beginning of the design cycle, a feasibility analysis of the model could be carried out using Relations and Constraints level in the Computation Domain, Timing Constraints in Time domain, Data Type Constraints in Data domain and Structural and Interface constraints in the Communication domain. This analysis can be easily represented in the Rugby model.

Tools for performance modelling and formal verification can also be represented in this framework. For example, a tool checking deadlock in a system may involve a model at the concurrent processes level in the Computation domain, at the causality level in Time domain, at the symbol level in the Data domain and at the inter process communication level in the Communication domain.

### 4.4: Unexplored Problems and Issues

In addition to the representation of well formulated problems, the Rugby model brings out a number of problems which have not been addressed adequately yet. Thus Rugby model not only helps in understanding the current state-of-the-art in design and design automation, it also points to gaps and provides direction for future research.

- 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 level models, which typically use abstract symbols for performance modelling, based on queuing 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.

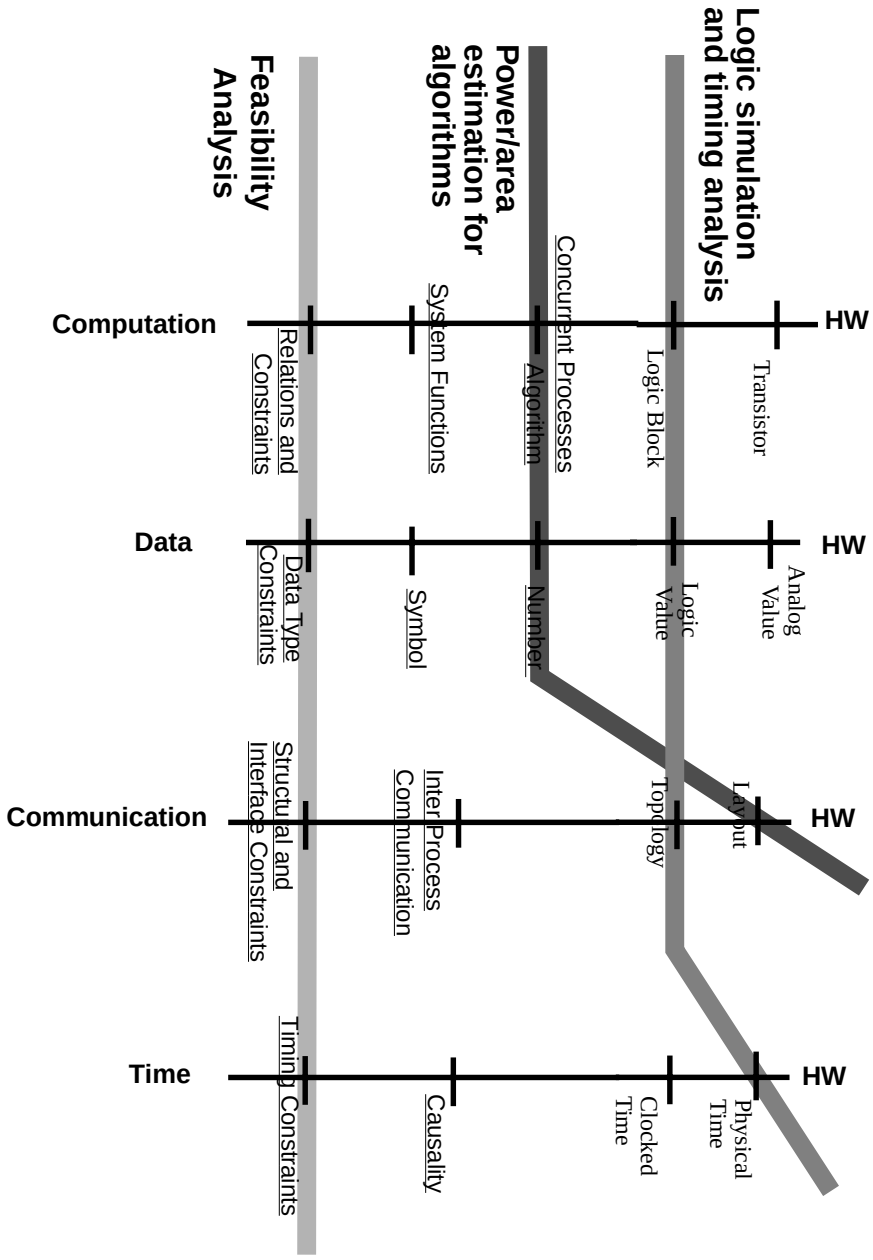


**Figure 4. Design Analysis Tasks in the Rugby Model**

- Transformation of system functions into concurrent objects indicated as functional partitioning in figure 3: In early design phases a system is best described in terms of use cases [8], scenarios [9], 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 [4]. 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.

## 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 for modelling 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 have shown how current tools for design manipulation at different levels can be represented.

A good framework not only helps in studying and understanding the current state-of-the-art but also points out gaps and areas for future work. Our framework highlights that there is a lack of appropriate representation of models at the system level and of tools for the refinement and analysis of models at the highest level of all four domains. Design methodologies can use this framework to define design documents and process steps which correlate abstraction levels in different domains in a specific way. For instance a methodology could select one domain leading the refinement process, e.g. the abstraction levels in the data domain could define process steps and each refinement of data would trigger a related refinement in the other three domains.

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 development and bring discipline in the electronic design automation area.

## 6: References


[1] Daniel D. Gajski and Robert H. Kuhn, "Guest Editor's Introduction: New VLSI Tools", IEEE Computer, December 1983, pages 11-14.

[2] Wolfgang Ecker, Michael Hofmeister, and Sabine 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, edited by Ronald Waxman and Jean-Michel Berge, Current Issues in Electronic Modeling, vol. 4, Kluwer Academic Publishers, 1996.

[3] Axel Jantsch, Shashi Kumar, and Ahmed Hemani, "The Rugby Model: A Framework for the Study of Modelling, Analysis, and Synthesis Concepts in Electronic Systems", *Proceedings of Design Automation and Test in Europe (DATE)*, 1999.

[4] Axel Jantsch and Ingo Sander, "On the Roles of Functions and Objects in System Specification", *Proceedings of the International Workshop on Hardware/Software Codesign*, 2000.

[5] James A. Rowson and Alberto Sangiovanni-Vincentelli, "Interface-Based Design", *Proc. of the 34$^{th}$ Design Automation Conference*, 1997.

[6] 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 EuroDAC 96*, Geneva, Switzerland, September 1996.

[7] Johnny Öberg, Anshul Kumar, and Ahmed Hemani, "Grammar-based Hardware Synthesis of Data Communication Protocols", *Proceedings of the 9th International Symposium on System Synthesis*, pp. 14 - 19, 1996.

[8] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Övergaard, *Object Oriented Software Engineering: A Use Case Driven Approach*, Addison Wesley, Reading, Massachusetts, 1992.

[9] Colin Potts, Kenji Takahashi, and Annie I. Anton, "Inquiry-Based Requirements Engineering", *IEEE Software*, March 1994.

[10] VSI System Level Design Model Taxonomy, VSI Reference Document, version 1.0, http://www.vsi.org/library/specs.htm, Virtual Socket Interface Alliance, October 1998.