Requirements, Primitives and Models for Systems Specification

César Augusto Missio Marcon, Ney Laert Vilar Calazans, Fernando Gehm Moraes

Pontifícia Universidade Católica do Rio Grande do Sul (FACIN-PUCRS)
Av. Ipiranga, 6681 - Prédio 30 / BLOCO 4 - 90619-900 - Porto Alegre – RS – BRASIL
{marcon, calazans, moraes}@inf.pucrs.br

Abstract

This paper presents a metamodel, called RPM, which is proposed to systematize and encompass many models of computation. RPM is based on the decomposition of a computational system model into its requirements, primitives and models of computation. A taxonomy for telecom computational systems is proposed and related to the RPM metamodel. The goal of the paper is to pave the way for a codesign CAD system with heterogeneous specification. Such systemic specification is to be mapped into one or more homogeneous descriptions.

1. Introduction

Computational systems are defined here as those formed by a composition of hardware and software. Today, many computational systems are implemented as *embedded systems*, often using a *system on a chip* (SoC) approach. The growing complexity of such systems leads to the increase of the abstraction level where activities like specification; design and validation are guided. The main goal is to render manageable the complexity of the design process of products based on these systems, by reducing time-to-market figures.

Computer aided design (CAD) systems congregate methods and tools that the designer employs to trace the design path from specification to physical implementation. CAD systems are based on an underlying set of *models of computation* (MOCs [9]) used for modeling, validation and synthesis stages of the design.

Understanding how MOCs are used, how different MOCs relate to each other and how MOCs related to computational systems are important issues in dominating the design process. In order to provide a framework for the reasoning about MOCs, this work proposes the RPM metamodel (Requirements, Primitives and MOCs). RPM is useful in the process of choosing MOCs to employ during systems modeling. It can also be used for reasoning about the relationship and interfacing between different MOCs. This work also introduces a new taxonomy for telecom computational systems based on a set of criteria derived from the RPM characteristics. The major goal of this work is to allow the construction of system level CAD systems capable to deal with heterogeneous descriptions of computational systems, where each subsystem may be modeled using several distinct MOCs.

This paper is organized as follows. Section 2 introduces the RPM metamodel proposal. Section 3 suggests a new taxonomy for telecom systems, and uses the telecom systems classification for a RPM case study. Section 4 presents conclusions and future work.

2. The RPM Metamodel

A *metamodel* is a formalism that represents relationships among models. Here, the interest is to represent the process of computational systems model construction. Figure 1 illustrates the RPM metamodel, comprising *modeling requirements* (or just *requirements*), *modeling primitives* (or just *primitives*) and *MOCs*. These concepts, with their relationship, define the construction process of *computational system models*. Each one of the concepts is further discussed in the next Sections. RPM captures the fact that a model of a system comes from the set of requirements that are fulfilled by primitives. These primitives are used to select existing MOCs or propose new ones that allow modeling the primitive's composition. Finally, MOCs separately or in conjunction with others serve to formalize the system classes features, allowing the computational systems modeling.
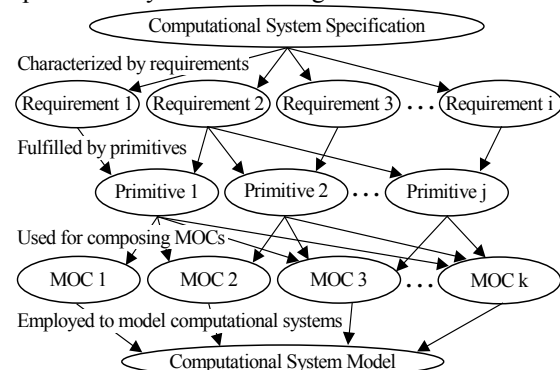

Figure 1 – Elaboration of Computational System Models

2.1 Modeling Requirements

Intuitively, a *modeling requirement* can be defined as a need, either to express intrinsic features, or to describe

classes of structures or behaviors often found in systems (e.g., the *concurrency* in a communication protocol).

This Section proposes a set of modeling requirements, grouped according to characteristics in two classes, *desirable features* and *constraints*. *Constraints* are the set of requirements that satisfy the need of expressing intrinsic features such as total circuit area. Constraints are strong requirements that limit the system implementation domain. *Desirable features* are requirements that express classes of structures or behaviors often found in systems. These requirements are our main interest since they are the bases of RPM metamodel construction.

Abstraction reduces the computational system design complexity. The designer employs several levels of abstraction, adding information, as the design is refined.

Modularity allows system description by breaking them in a set of subsystems, based on: *high cohesion* and *low coupling*. Cohesion is acquired by the grouping of related structures and coupling is given by intermodule dependency.

Hierarchy is the capacity to repeatedly break subsystems into its constituent parts, each part being a subsystem in itself. Hierarchy is strongly related to abstraction and modularity concepts. If a (sub)system is formed by a set of non-overlapping subsystems these define a *hierarchy level*. Further subdividing each subsystem leads to another set of subsystems, defining a *lower* level of hierarchy.

Concurrency is the capacity of representing flows of parallel events. This work associates flows of parallel events with the *process* concept, defined in Section 2.2. Concurrent systems are those with more than one flow of parallel events, which can mutually influence each other.

Communicability is the capacity of exchange information among subsystems or with the environment.

Dynamicity is the capability of a system to modify its behavior. Dynamicity involves creation and elimination of memory and processing resources. In general, dynamic behavior is associated with software systems, while hardware systems are characterized by static behavior.

Stateability is the capacity of a system being represented by a set of "situations", called *states*. A state is associated with a time interval, the period that the system is in one of its possible distinct situations. To change this situation is called *to modify the state of the system* in which it will remain during another time interval.

Determinism expresses a guarantee that starting from a given state a system will always give the same response to the same stimuli. Determinism allows foreseeing sequences of output signals if sequences of inputs are known. Although a deterministic behavior is generally desirable, the capability of expressing non-determinism can potentially increase the power of models.

Real time operation establishes a maximum response time to any or to a significant subset of inputs to the system. This work subdivides real time systems (RTS) in 3 classes. A *non-real time system* (NRTS) is one where execution time is not dictated by input events. They operate at a given rate and provide results whenever they are ready. In essence, variation of its execution time may not have side effects on the expected external system behavior. *Non-critical real time systems* (NCRTS) are systems whose operation time is dictated by input events, but the execution time is not relevant for the system available resources. These systems can perform several operations until they need to answer to some input event. *Critical real time systems* (CRTS) are systems whose operation time is dictated by time events and the time constraints are near to technological limits. The performance of the designed is a determinant factor.

An *exception* is a deviation of the most common expected system behavior. Design errors or relations between the system and the environment, such as external interrupts may cause exceptions. Exception handling is an important requirement when dealing with non-determinism.

Reliability implies a precise behavior during operation time [8]. Average reliability for communication systems are achieved by using mechanisms to avoid information loss (e.g. unbounded FIFOs). High degrees of reliability can be achieved by the use of redundancy.

2.2 Modeling Primitives

A *Modeling primitive* is a basic element that isolated or together with other primitives is capable of satisfying the needs of system modeling. The extraction of necessary and sufficient primitives is a challenge, and is a basis for sound MOC construction. Excessive number of primitives makes MOCs too complex. Using less primitives than necessary eliminate the models capacity to represent systems behavior.

A *value v* is a graphical representation that only has meaning together with other elements. A *unit u* is a representation that allows characterizing its associated elements (e.g. bits). The composition of value and unit allows characterizing primitives both qualitatively and quantitatively. Sets of composition and/or derivation rules enable the specialization of new primitives. In this context, the process of building primitives is called *specialization*. Primitives are hierarchically defined in terms of composition relations, derivation or dependence as stated in Figure 2.

Composition relations contain information to produce a new primitive. *Derivation relations* tell that the derived primitive is a more specialized case of the element that it derives. *Dependence relations* carry the information of the set of primitives, which depends on a new primitive creation.

A pair (*v*, *u*) composes *physical primitives* representing some physical aspect of the computational system, such as speed. The study of these primitives for MOCs construction is not treated here, since this work focus on behavioral models. Although *time* is a physical primitive, it is not treated so, because RPM uses it as an ordering element for events and their derived or composed primitives. *Behavioral primitives* include all primitives derived, composed or dependent on the mathematical elements, or another behavioral primitives previously defined and without physical information.
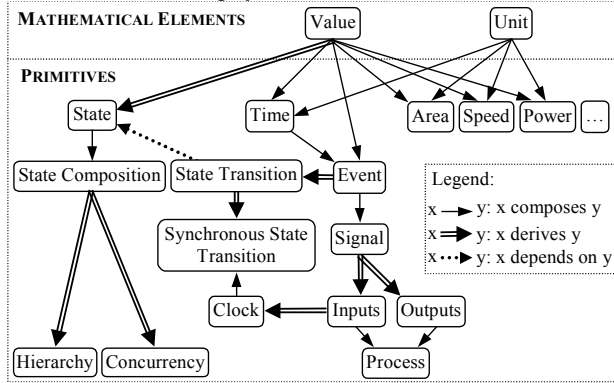
Figure 2 – Constructive process of primitives

The *time t* is a primitive composed by (*v*, *u*) (e.g. 4 ns). The time primitive allows establishing a total order of elements that are referenced by it.

State is one of the possible situations that a system can be. In [4], the authors formalize the concept of states as equivalence classes of processes. Nevertheless, this concept is unsuitable to derive state composition, hierarchy and concurrency concepts. In the context of this work, a state **q** is a label that will be treated as a primitive derived from (and identified by) a numerical value.

Event is a primitive that carries the information of a value *v* in an instant of time *t*. This work adopts the formal analysis proposed in [4] that represents an event as a pair (*t*, *v*). Given a set of all instants of times *T* and a set of all values *V*, an *event e* is defined as an element of the Cartesian product *T* x *V*.

A signal s is a primitive composed by a set of *n* events that define, at every moment, the signal value. Hence, a signal can be analyzed in a behavioral form as an entity on which a sequence of events occurs. *Inputs i* and *outputs o* are primitives derived from *s*, with $i = \{e_1, e_2, ..., e_l\}$, and $o = \{e_{l+1}, ..., e_n\}$. The set of all inputs *i* is represented by *I* and the set of all outputs *o* is represented by *O*. A *clock* is derived from the input primitive and is characterized by being a signal composed by events with periodic time tags.

A *process P* is a computational system behavior that maps the input alphabet to the output alphabet. *P* is defined as a set of relations between inputs and outputs.

State transition is a primitive derived from the event primitive and dependent on the state primitive. In

asynchronous systems, events and transitions are closely related, since the occurrence of an event can cause a state transition. However, in synchronous systems, the occurrence of events will only be evaluated at the occurrence of a synchronism signal event. The essential difference between events and transitions is that not all events cause a transition, but all transitions are caused by some event(s).

Let *Q* be the set of all states of a system, $Q = \{q_1, q_2, ..., q_n\}$, and let *Φ* be the powerset of *Q*. The *state composition C* is a subset of *Φ*. Given that the system behavior can be partially represented by *Q*, two behavioral primitives can be derived from *C*: hierarchy and concurrency. In a 2-level hierarchical representation, the pair (q_{1i}, q_{2j}) implements *C*, where q_{1i} is the i-th state of level 1 and q_{2j} is the j-th state of level 2. Generalizing the concept, in an *m*-level hierarchy, *C* is implemented by one *m*-tuple (q_{1i}, q_{2j}, ..., q_{mk}). A system is named *hierarchical* if at least one state is hierarchical.

Under the *C* point of view, a concurrent system is perceived in the same way as a hierarchical system. However, in purely hierarchical systems (without concurrency), each one of the *m* positions of *C* represents one level of the hierarchy, where all the states are composed by sets of states. The set of states that belongs to the state of the lowest hierarchical level is empty. In purely concurrent systems (without hierarchy), there are no transitions between states that are part of each one of the *m* distinct positions of an *m*-tuple. Thus, each position of the tuple is composed by an unconnected set of states.

2.3 Requirements and Primitives

Table 1 represents the relationship between primitives and requirements. Table 1 is partially outlined in Figure 3.

Table 1 – Relation between requirements and primitives

Requirements / Primitives	Determinism	Abstraction	Modularity	Hierarchy	Concurrency	Communicability	Dynamicity	Stability	Real time operation	Exception handling	Reliability
Time	■						■		■		
State				■							
Signal		■	■		■	■		■		■	■
Clock	■						■	■	■		
Process	■				■		■		■	■	
State Transition	■						■				
Hierarchy		■	■	■			■			■	
Concurrency			■		■				■		■

2.4 Models of Computation

Single MOCs are those created by the composition of primitives only. Composite *MOCs* are those created by single MOCs composition or derivation. Any MOC is

produced by the aggregation of specific primitives, providing a formal structure for modeling a significant amount of computational systems classes. Figure 3 presents a few single MOCs, some composite MOCs and the relationship between MOCs and primitives.
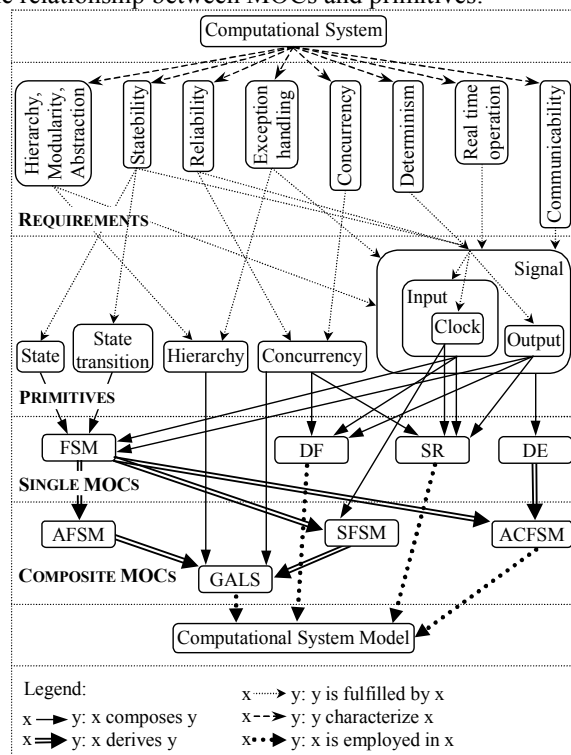
Figure 3 – Metamodel example

The set of single MOCs illustrated in Figure 3 is composed by *finite state machine* (FSM) [5][6][10], *data flow* (DF) [2][6][9], *synchronous reactive* (SR) [1][4] and *discrete event* (DE) [4][6]. The set of composite MOCs illustrated in Figure 3 is composed by *asynchronous FSM* (AFSM) [9], SFSM, *abstract codesign FSM* (ACFSM) [9], *global asynchronous local synchronous* (GALS) [9]. Other single MOCs like *process network* (PN) [7], or composite MOCs like *hierarchical FSM* (HFSM) [10] and *Control/Data Flow Graphs* (CDFG) [7] are not showed.

The MOC representation capacity is affected by its degree of specialization that reduces the amount of express systems that it can. On the other hand, the less abstract, the more expressive it will be, intensifying its representation capacity. Each computational system interacts somehow with its *environment*. The environment interaction establishes an association with aspects such as the system processed information flow, the system operation (synchronous or asynchronous) and the system behavior. The synchronous and asynchronous operations determine different models to subsystem intercommunication.

Reactive, interactive and transformational are system classifications according to the behavior criterion [1]. *Reactive systems* react to the environment stimuli

returning new stimuli to it. A typical MOC used to model this behavior is the SR model [1]. *Interactive systems* are constantly exchanging information with the environment. However, in contrast with reactive systems, the main agent of the interaction is the system instead of the environment. *Transformational systems* are characterized by the low interaction with the environment. These systems compute the output values from the input values and then stop.

Time primitives enter in the composition of almost any MOC. Time can be modeled in discrete or continuous forms. The continuous form takes the information from all the signals at every time instant, in opposition to the discrete form, where only the instants of time with associated events are considered.

There are two modeling classes based on *MOCs composition*. The first is *homogeneous modeling*, where it is necessary only one MOC to obtain sufficiency and expressiveness for the modeling. This class is adequate for systems composed by a single subsystem, or where the behavior of all the subsystems is similar. The second is *heterogeneous modeling*, where it is necessary the composition of more than one MOC. Systems with low complexity can often be modeled homogeneously, due to the uniformity of its subsystems. When complexity grows, heterogeneous modeling becomes a better approach.

2.5 Systems Modeling Approach

Several composite MOCs are proposed in the literature [2][3][5][10]. These MOCs have high potential to express some computational systems that are not easily expressed by single MOCs. This paper proposes a *Heterogeneous System Modeling with Homogeneous Subsystem Modeling* approach (HSM^2), which can fulfill the needs of many computational systems. HSM^2 is a method for systems construction, which enables the use of simple and expressive MOCs for subsystems, plus a set of subsystem interoperability MOCs. This approach can be divided in two main steps. The first one is the choice of a representative set of MOCs for each subsystem. The second one is the interoperability MOCs definition, not addressed here. The advantage of this approach is the complexity reduction, due to the simplicity of each subsystem underlying MOC. A drawback is found in interoperability MOCs, due to the fact that some MOCs have opposite characteristics that many times mandate the improvement of the interface to enable lossless intermodule communication. This approach will be exemplified further in Section 3.4.

3. Telecom Systems

Telecom systems can be defined as those characterized by a behavior comprising mostly the transmission and reception of information. The exchange of information is

the ultimate goal of such systems. This definition can be useful for the telecom systems understanding, but it is not useful to extract information for their modeling, due to its high abstraction degree. It is thus necessary to propose classification criteria for these systems to allow construction of representative MOCs for each class.

3.1 Classification Criteria

The analysis of several telecom systems provided a basis to propose a set of criteria that characterize such systems. The criteria set is expected to be sufficient to define suitable MOCs to a large amount of systems. The set comprises six main criteria, explained next.

1. **Environment interaction** – similarly to MOCs, telecom systems can be classified by its interaction with the environment in transformational, reactive or interactive. For example, reactive systems verifiability is usually easier than interactive systems verifiability, due to the fact that reactive systems are, in general, in a stable situation when it receives inputs.

2. **Real time operation (RTO)** – several telecom systems are characterized by its RTO. An example of a NRTS is an Internet service like the transmission of an e-mail message. NRTS allow several degrees of freedom for implementation. They can be implemented in hardware or software without problem. An example of NCRTS is a telephone system, interacting with the user, where the associated time is in the order of seconds. An example of CRTS is an ATM switch, where the switching of the data must occur at nanosecond rates.

3. **Encompassed Layer(s) of the OSI-RM** – the OSI layers are associated with issues like execution time and environment interaction. The higher OSI-RM systems layer, the more abstract the primitive of the underlying MOC can be. Lower OSI-RM layers have to operate at high speed, and in most cases this implies CRT systems.

4. **Complexity** – this is a mostly subjective criterion, but it gives an idea of how difficult the problem and how complex it's underlying MOC must be. Complexity is a criterion that is directly related to modularity, hierarchy, abstraction, concurrency and communicability.

5. **Reliability** – most telecom systems need to operate with an average reliability degree, but for special cases high reliability degree is necessary.

6. **Synchronous/asynchronous operation mode** – many telecom applications are characterized by synchronous communication, like SDH or E1 systems. These systems are better modeled by synchronous MOCs. Other telecom system applications are characterized by asynchronous communication, similar to ATM systems or the Ethernet protocol. Systems like these are better modeled by MOCs with asynchronous attributes.

3.2 Telecom Classes Proposal

This Section summarizes in Table 2 a classification for telecom systems according to the above criteria.

Table 2 – Summary of telecom systems classification

Class \ Criterion	1	2	3	4	5	6	7
Environment interaction	I	I	I	R	I, T	*	*
Real time operation	CRT	CRT	CRT, NCRT	NRT	CRT, NCRT	NRT	CRT
OSI-RM	L	L	L, M	H	H	*	*
Complexity	L, M	L, M	H	H	H	M, H	H
Reliability	M	M	L	L	M	M	H
Operation mode	S	A	*	*	*	S	*

Legend:
R - reactive; **I** - interactive; **T** - transformational; ***** - undefined
S - synchronous; **A** - asynchronous; **H** - high; **M** - middle; **L** - low

Typical circuits of these classes are:

Class1 - SDH or E1 protocol interfaces;
Class2 - High speed asynchronous circuits as ATMs;
Class3 - Compression/decompression of images;
Class4 - Circuits that interact with the users of the communication media such as occurs in TV;
Class5 - Circuits that interact with other circuits of the communication media;
Class6 - Computer connection systems;
Class7 - Some circuits used in military applications.

3.3 Relation Among Classes and RPM

Telecom subsystems must be associated with the MOC that expresses its behavior. This can be obtained by two approaches. The first starts from the previous knowledge of the system specification underlying MOCs. These MOCs can be obtained through the case studies of implemented systems; The second starts from the extraction of the associated subsystem requirements obtaining the primitives that fulfill them, consequently obtaining the MOCs composed by those primitives, such as detailed in the RPM construction. This work adopts the second approach. Due to the complexity of telecom systems, two levels are inserted between the computational system and the requirements of RPM, as it is illustrated in the grey area of Figure 4. The classification criteria give the relation between telecom classes and MOCs.

RPM applied on telecom classes enables the deduction of suitable MOCs for each classified telecom system. These MOCs are obtained through the classification criteria summarized in Table 2. This Section illustrates a Class1 example. The Class1 criteria can be related to a set of requirements. *Interactivity* establishes an association with *communicability*. *CRT* operation is related to *RTO* and *concurrency*. *Low layer of OSI* is associated to *communicability* and *statebility*. *Low or middle complexity*

leads to primitives with low or middle degree of abstraction, communicability, concurrency, and hierarchy. Synchronous mode operation is related to determinism.
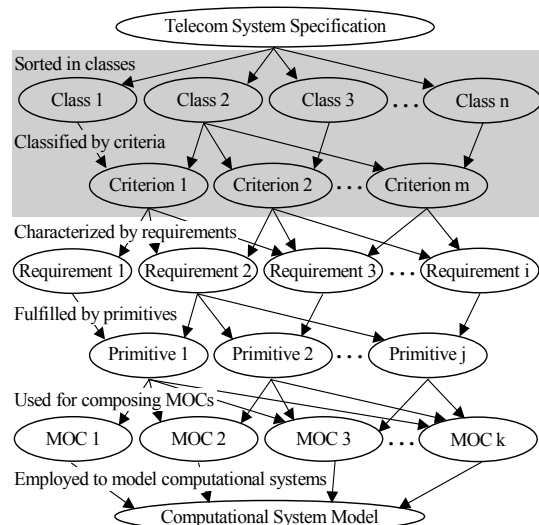

Figure 4 – Construction of Computational System Models

Associating these requirements with primitives, as presented in Table 1, it is possible to verify that hierarchy and concurrency primitives are less important primitives to compose MOCs for Class1 systems. On the other hand, signal and state primitives are very important. RPM shows that MOCs can be deducted from primitives, as outlined in Figure 4 and Figure 5. DE and FSM are adequate for an abstract representation of this problem class. DE includes the notion of physical time, what makes it adequate for the hardware modeling [5]. FSM is adequate for the sequential modeling and for systems with low complexity and few processes, because it does not intrinsically support the hierarchy and concurrency requirements [1].

3.4 HSM² Application Example

This Section presents an application example of HSM² approach for telecom system. This example is a system composed by 3 subsystems: *User*, *Control* and *Comm*. The User subsystem implements the system interface that interacts with the user. The Control subsystem implements the core functions. Comm subsystem implements the communication with other systems. In this example, each subsystem is being modeled by its MOC, which appears between parentheses in Figure 5.
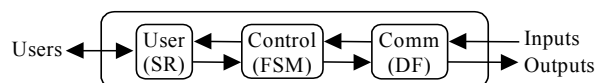
Figure 5 – An example of HSM² approach

The absence of this approach would probably force the use of a more complex MOC to fulfill all subsystem requirements with only one MOC. This MOCs complexity usually leads to a reduction of the system performance.

4. Conclusions and Future Work

It is essential to obtain MOCs that are at the same time representative and simple enough to avoid unnecessary computational costs. This may occur when the primitives of the MOC force the over-specification of a system. The RPM metamodel coupled with the HSM² approach may assist in the design of computational systems, reducing the system complexity and increasing the expressiveness, through the association of the subsystems of a system, with the requirements, primitives and MOCs. The telecom classification allows a better choice of MOCs to use in modeling such systems and to better decompose them.

The RPM Metamodel and the HSM² approach are the more abstract levels of an ongoing effort to implement a codesign system. This system is planed to have as design entry heterogeneous language specifications with different underlying models. During the design flow, the specifications are converted into intermediate descriptions guided by the designer. These descriptions will represent the successive refinements of the system model using different languages, each one designed to support different MOCs. The interaction with the intermediate languages is to be achieved by the support of the heterogeneous MOCs given by the HSM² approach.

5. References

[1] A. Benveniste et al – The Synchronous Approach to Reactive and Real-Time Systems. Proc. of the IEEE, 1991.

[2] B. Bhattacharya et al – Parameterized dataflow modeling for DSP systems. IEEE Trans. on Signal Proces., 2001.

[3] J. Buck et al – Heterogeneous modeling and simulation of embedded systems in El Greco. CODES, May 2000.

[4] S. Edwards et al – Design of Embedded Systems: Formal Models, Validation and Synthesis. Proc. of the IEEE, 1997.

[5] A. Girault et al – Hierarchical Finite State Machines with Multiple Concurrency Models. IEEE Trans. on CAD of Integrated Circuits and Systems, June 1999.

[6] A. A. Jerraya et al – System-Level Synthesis, pp.103-136, Kluwer Academic Publishers, 1999.

[7] L. Lavagno et al – Embedded System Codesign: Synthesis and Verification. Kluwer Academic Publishers, 1995.

[8] R. S. Pressman – Software Engineering: A Practitioner's Approach. McGraw-Hill, Fifth Edition, 2000.

[9] M. Sgroi et al – Formal Models for Embedded System Design. IEEE Design & Test of Computers, April 2000.

[10] V. Sklyarov – Hierarchical Finite-State Machines and Their Use for Digital Control. IEEE Trans. on VLSI Syst. 1999.