Towards a viewpoint oriented design methodology for Multi-processor Real-Time Systems

L. Zhang, J. van Katwijk
Delft University of Technology
Faculty of Mathematics and Computer Science
132 Julianalaan, Delft The Netherlands
Fax: +31 15 78 71 41
Tel: +31 15 78 64 11

May 27, 1994

Abstract

In this paper we describe a design methodology for the development of (potentially multi-processor) real-time system. Our proposed methodology differs fundamentally from current methodology and serves to help manage the complexity of massive-intensive systems. One of the distinguishing aspects of this methodology is in its ability to express timing constraints and verify to what extent such constraints are met. A second distinguishing aspect of this methodology is to develop a system according to *five* views of systems. In particular, the problems concerned with the transformation of the specifications into the parallel programs are addressed.

The proposed methodology is applied to develop an onboard generator for generating transfer frames, complyannt to the CCSDS Recommendations [34], [35]. This onboard generator is implemented on a transputer network [1], [2].

Index Terms Real-time systems, specification, performance evaluation, parallel processing, CCSDS, transputer.

1 INTRODUCTION

Many tasks performed in systems, such as those found in nuclear power plants, process control and spacecraft applications have stringent timing constraints. The real-time contraints are hard, failure to meet them might cause a catastrophe. Although computer speed has increased by several orders of magnitude in recent decades, the demand for computing capacity increases at an even faster speed. Consequently, the required processing power for many real-time applications still cannot be achieved with a single processor system and these applications need to be implemented on multiptocessor systems.

On the other hand, advances in VLSI technology have made it possible to construct high-performance parallel computers with large numbers of processing elements. This is especially true with the appearance of transputer systems [1], [2], where each transputer is constructed as a single device with private memory, processor, and communication links. Due to their regular structure, transputers can be linked together to give high performance systems with arbitary topologies. Transputer networks therefore provide a favourable target environment for designers of complex real-time systems.

A real-time system distinguishes itself from other systems by the explicit involvement of the dimension time and the dependability of the correctness of results as a function of time. According to many authors,[3], [65], [67], [69], [72], [73], [87], real time systems are subject to fundamental user requirements, such as correctness of the system, taking into account timeliness and simultaneity, a high degree of predictability, of robustness.

Distributed and parallel target environments, in which timing behavior depends on many factors including task allocation, scheduling, and communication, complicate the analysis of timing requirements. Thus, approaches to cope with the analysis of time during all phases of real-time development are needed.

It seems fair to say that current methods that aid in the development of (multi-processor) real-time systems are still underdeveloped [3], [65], [69], citekn:gnu64. Numerous methods have been proposed for the design of such systems, however, none of these methods correxctly addresses all kinds of user requirements.

In our research we did not find a single method that covers all phases in the development cycle. As an example, consider Jackson System Design [24], [63], MASCOT3[?], RTSA[26], Statecharts [27] and HOOD[29], well known methods for the development of real-time systems. JSD does not provide means to express the decomposition of the system until the process level, MASCOT3's graphical notation for process decomposition does not show control flow. HOOD and other object models have problems with the poor execution time performance [38], which by the way is not a problem with the object abstract abstraction itself, but ususally follows from an inefficient implementation of access to objects. Statecharts are appropriate for behavior analysis. but the underlying method fails to address performance analysis. Finally, RTSA, a variant on Structured Design[26], deals with decomposing a system into modules, but has limitations in its ability to support design of concurrent system.DARTS [75] is a software design method for distributed real-time application, however, it not concerns with the schedulability analysis.

Therefore, the objective of our works is to develop a *single* design methodology that covers *all* phases of the life cycle while ensuring that the specific real-time system requirements of the software will be met, even on parallel architectures as target. A major difference between current methodologies and our methodology is that we freely use a number of different methods in the process of systems development. For each stage (or view as we will see later), one or several specific methods are candidate for use. These methods are selected according to the characteristics of real-time systems and the requirements of each particular stage. Whenever required. we extend particular methods in order to satisfy the needs for the particular view we are dealing with.

Another major difference between current methodologies and our methodology is that we emphasize an integration between informal methods and formal techniques, exploiting the advantages of informal methods and formal methods.

This paper is organized as follows. In section II we outline the main aspects and characteristics of the proposed methodology and compare it to other methodologies. In section III we discuss issues concerning the application of multiple views within phase oriented development. In section IV we present a case study, the detailed development of a simulator for generating the transfer frame of CCSDS Recommandations. In that section we furthermore describe how dynamic allocation and reconfiguration requirements are met in an implementation on a network of transputers. Finally, in section V we summarize the features of our methodology and discuss future work.

2 CHARACTERISTICS OF OUR METHODOLOGY

2.1 Different views on a system

An acceptable real-time design methodology must synthesize the different *views* that may exist on a system, and must ensure that an orderly series of steps covers all aspects of the requirements and the design aspects. Hence, we propose a methodology that explicitly emphasizes the use of (five) complementary views on the system as a starting point for modeling and design. The views are: the *environmental view*, the *functional view, the behavioral view*, the *performance view* and the *material view*. Each of these views will be discussed in detail below.

The environmental view: Embedded computer systems have to react quickly and correctly to complex sequences of external events. The entities that produce these (external) events are collectively named "environment". Since the behaviour of the system is strongly coupled to the behaviour of the environment, an accurate analysis and description of the behaviour of the environment is important in this application domain. We propose a view of the environment of system as part of the development of the system. In building this view, we are concerned with:

- the identification of objects in the environment of system;
- the determination of operations and events related to these objects;
- the description of the behaviour of each object;
- the specification of the timing constraints of the output of each object.

There is common agreement that object-oriented models closely match our model of reality. Typical object-oriented notations, such as provided by HOOD, and notations belonging to a variety of formal techniques (e.g. RT-ASLAN [58], Real-Time Logic (RTL)[59], ESTEREL[29], LDS[30], LOTOS) are suitable for describing the environment of the system. HOOD is typically used for the identification (and informal description) of objects in the environment, subsequently, formal techniques are suitable for the behavioural description of each of the elementary objects. The choice of the particular formal technique depends on the application.

Specification of timing constraints on output signals can be given in a variety of notations, we prefer a simple specification notation that, in a later stage, can be embedded within other notations. In this notation, two kinds of constraint are identified, *periodic* and *sporadic* ones. A periodic constraint requires some action to be executed at fixed intervals while some state predicates are true. A sporadic timing constraint requires some action to be executed before a specified deadline elapses after the occurrence of a certain event.

Timing constraint of < output signal > on system implementation is
While < event >
demand to execute < action > and finish to execute < action >
WITH period = < time >, deadline = <time>

The proposed syntax of a sporadic timing constraint is:

Timing constraint of < output signal > on system implementation is
While < event >
demand to execute < action > and finish to execute < action >
WITH deadline = <time >, separation=< time >

The separation parameter in a sporadic timing constraint specifies a lower bound on the length of an interval separating two successive occurrences of the triggering event. The purpose of the separation parameter is to prevent any source of sporadic requests for computation to hog the system.

The functional view: The *functional* view captures the static structure of the system, it addresses questions on functionality of the system, i.e. the input and output, what the subfunctions are, and how these functions are combined.

We propose a decomposition to be performed by using an extension of SADT. SADT [36], [50] is a graphical language used for explicitly expressing hierarchical and functional relationships among any objects and activities, and has been successfully applied to functional system decomposition, Unfortunately, SADT is not particularly suited to express the types of communication and synchronization between different functions. Therefore, we propose an extension that allows three types of relations between functions, to be expressed by lines of specific types in the diagrams [64]:

- Synchronization by event, represented by the line of figure 2.1.

  ....

- Transfer of information by state variable, represented by the line of figure 2.1.

- Transfer of information through ports, represented by the line given in figure 2.1.

  Apart from the so-called structured methods, a wealth of formal specification notations exist, we mention VDM and Z[86]. In [88] the integration of formal methods ans structured methods in software development is addressed in detail.

The behavioral view: The *behavioral* view captures the dynamics of the system, i.e. it captures under which conditions the functionality is performed.

On each level of the activity hierachy, the control activities must be presented, controlling that particular level. These controllers are responsible for specifying when, how and why things happen as the system reacts over time.

We propose a combination of Statecharts/Real-Time Logic[59] to specify the behavior of a system. Statecharts, extensions to conventional Finite State Machine (FSM), make it even easier to model complex system behavior unambiguously. Conventional state diagrams are inappropriate for the behavioral description of complex control, since they are flat and unstructured, are inherently sequential in nature and give rise to an exponential blow-up in the number states. Statecharts do overcome these problems by supporting a repeated decomposition of states into substates in an AND/OR fashion, combined with an instantaneous broadcast communication mechanism. A rather important facet of these extensions is the ability to have transitions leave and enter states at an any level of decomposition.

A major drawback of statecharts is their underlying assumption that a transition between two states does not take any time. Statecharts are therefore less suited for expressing internal timing constraints of a system. As a remedy, Real-time Logic is used for the specification of the condition that guards the transition.

The performance view: Performance refers to system responsiveness: the time required to respond to specific events, or to the number of the events processed in a given time interval. Other than for e.g. information systems, and even for soft real-time systems, in hard real-time systems, performance issues are correctness issues.

In today's embedded computer systems, performance requirements are a major concern. Due to the time criticalness of such an application, performance analysis and optimization becomes prominent. Hence, we propose a performance view, in which the performance of the artefacts under construction is critically inspected, and where the performance of the resulting system is kept under control within the development process. In building this view on the system, analysis and simulation models of the system are developed. The function of these models is to help to determine performance bottlenecks in the system and to assist in the selection of the final hardware configuration. The basic technology is simple, rough estimates of processing times and communication times for elementary functions are computed. These estimates are tallied to verify wether the complete system fits within user defined multi-processors or on (hypothetical) multi-processors and can meet time constraints. An analytical model can be constructed, which is based on this data. Such a model provides possibilities for a computer-based simulation, gauging the effects of alteration or extension to system design. In order either to simplify the model and make it more usable, we use a precedence graph [40], [6], model omitting needless details. These graphs for the basis for performing analysis through queueing networks [10], [39], [48] and simulation can be performed. Transformation from a functional diagram to a precedence graph (software partition [43]) deals with the process of defining tasks from given functional diagrams. Within distributed systems of the kind we are looking at, each processor has only limited local memory space and a restricted CPU throughput capability. If local memory space is not large enough to accommodate a particular task, that task needs to be broken down into smaller pieces. Also, given the throughput capability of a processor, if the arrival rate of an input type exceeds its service rate, its processing requirements need to be partitioned into multiple tasks and more than one node must share the processing responsibility.

The material view: The *material* view aims at supporting the process of fitting the specification onto a particular target hardware environment. A developer must decide what hardware should be used and how it is used to implement the specification. The material view provides means for evaluating particular configurations with respect to a particular application. Such a view can be represented by a pictorial representation of the system showing how the hardware is configured and how the tasks are implemented.

The number of processors is almost always smaller than the number of tasks, processors have to be shared among processes. Rules must be established for allocation and scheduling of tasks (or functions), while these scheduling policies themselves also should be implemented. Since task allocation and scheduling is part of the overall software engineering methodology, it should support the system objectives. One of the most critical system performance goals of a real-time application is to satisfy the response time requirement. Another important aspect is the communication and synchronisation between tasks. Real-time communication must be predictable in satisfying message-level timing requirement. The implementation not only ensures the logical correctness of a communication, but also timing correctness.

2.2 Methodological aspects

Common methodologies emphasize development to be structured in several phases. We do not share this view, rather than sticking to different phases, we emphasize on the different views on the system.

Of course, embedding the multiple-view approach within classical phase-oriented development models is possible, and even rather straight-forward. Depending on the characteristics of the development organization and the estimated size of the resulting product, a particular phase model can be selected. However, in those projects performed by ourselves, we have been using an evolutionary approach.

3 Performance analysis and Implementation

One of the most important views on the system under development, is the view in which performance analysis and schedulability check are central. Performance analysis is a quantitative form of analysis on software design, in order to model the behaviour of the system when external workloads are applied to it.

Schedulability analysis is a qualitative form of analysis, in which it is investigated whether or not particular schedules are indeed deasible within the target environment.

3.1 Performance models

Three analysis strategies guide the formulation of a performance model [83], [39], [84]. In early stages of the development process, the purpose of software performance engineering is to identify development plans that lead to software, meeting its performance objectives and, when multiple alternatives exist, that might provide data for the selection of the most desirable alterative. At this stage, designers focus on high-level decisions rather than implementation details. It is suggested that we use models that can be constructed quickly and that enable rapid evaluation of alternatives.

Performance studies in early stages of development seldom have precise input data. In early stages, both a best-and-worst-case analysis are needed to identify potential risks. If best-case predictions show that objectives will *not* be met, developers must seek feasible alternatives before processing. If the worst-case predictions produces satisfactory performance predictions, it seems that the models applied so far are indeed feasible, so that development can continue with its next phase.

Usually, the results are somewhere in between, in which case software performance engineering methods need to be applied to identify and focus on critical components to obtain more precise data and to monitor and manage critical-component performance as the system evolves. In these cases we need explicit performance models.

A performance model is an abstraction of the real system, developed for the purpose of gaining greater insight into the performance of the system, whether the system actually exists or not. This abstraction may be in the form of a mathematical model or a simulation model. In order to construct and evaluate performance models, we need several types of data [83], [84], [39]:

Performance requirements: Specific, quantitive performance requirements for the system must be defined.

Behavior patterns and intensity: This requires identification of the distinct types of events that occur as well as their intensity. Intensity is defined either interms of the arrival rate of each type of event or the number of concurrent users (or external entities) that interact with the system and the frequency of their requests (events). Specification for both steady-state and worst-case behavior are needed.

Software descriptions: The operations that provide responses to the events must be described. The level of detail increases as the software evolves.

Execution Environment: The execution environment is described by specifying the hardware devices and (significant) software service routines required for execution and the service rate for each of them. Any other work that may introduce resource contention delay must also be included in the execution environment.

Resource usage estimates: Resource requirements for operations, in terms of processor demands, I/O, and (later) memory requirements must be quantified. These provide the estimated number of sevice requests per device or sevice routine and the amount of service needed for each operation.

Based on precedence graphs [6], [40], queueing networks [10], [39], [48], and simulation techniques are used to evaluate performance of parallel programs.

The steps of performance analysis are as follows:

1. evaluate execution time, communication time and activation time of all elementary functions, related to a hypothetical multiprocessor on basis of the behavioural diagrams for those functions.
2. transform the function diagram into precedence graphs.
3. evaluate this transformation. If performance requirements are met, continue. If some performance requirements are not yet met, change the type of processor, change the number of processor or redo the functional analysis for a further decomposition of the function.
4. evaluate various strategies of allocation and scheduling to obtain the good performance of system.

3.2 Elementary execution time prediction

Methods for predicting the execution and communication time for real-time systems aim at finding best and worst execution times of elementary functions. In particular. the approach of Shaw [66]. [79], [81], aid in predicting such execution times. The method of Shaw is based on the following steps:

1. decompose a statement into primitive basic components (atomic blocks), as defined by the (language dependent) timing schema for the statement.

2. predict the implementation size for each atomic block. i.e. the code prediction.
3. determine the execution times of the atomic blocks from the times of machine instructions produced by the implementation.
4. Compute the execution times of statements. using the times of the atomic blocks and timing schema for the statement.

3.3 Transforming functional diagrams to precedence graphs

The transformation from a functional diagram to a precedence graph (software partition[43]) deals with the process of defining *tasks* from given functional diagrams. The transformation maps a given set of logical modules, reflecting the user's point of view, onto a set of software tasks, reflecting the implementor's view of the system.

The criteria for deciding whether a function should be a separate task or grouped with other function into one task are following [75]:

- dependency I/O
- user interface dependency
- periodic execution
- time critical function item computational requirements
- functional cohesion
- sequential cohesion

In figure 1, functions F1 and F2 are sequential, we combine them into one task task1, as well functions F4 and F5 are combine into task2, we obtain a precedence graph as showin Figure 1(b).

3.4 Verification of schedulability

When the functional diagrams are transformed into a set of tasks, the acceptability of this transformation needs to be decided. This evaluation is called Schedulability Analysis or schedulability check or verfication of schedulability.

A task or a task set is said to be schedulable if it meets all its deadlines. Guaranteed schedulability analysis attempts to determine whether a task or task set will be schedulable under a given condition. It emphasizes the predictability of the timing behavior of system. Usally, schedulability analysis concerns with determing if the timing requirements of a set tasks under a given set of well-understood scheduling algorithms.

Liu and Layland [90] showed that Rate Monotonic and Earliest Deadline scheduling are optimal static and dynamic priority scheduling algorithms for scheduling periodic tasks on a single

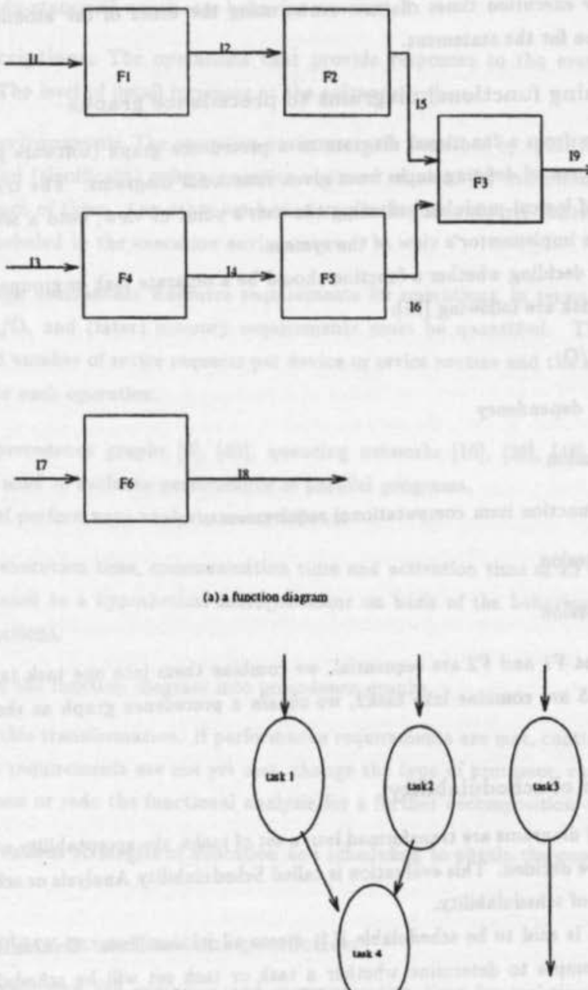


Figure 1: precedence graphe of a function diagram

processor system.

In multiprocessor systems, various scheduling algorithms have been proposed, however, none can be used in general case. A major problem is that most of the multiprocessor scheduling problems that have been studied so far rely on too many assumptions which do not hold in reality.

The Schedulablity check uses an earliest-deadline-first strategy to compute a feasible solution in which time constraint and a given precedence constraints are enforced. First, we check to see whether each processor is idle or not. If some processor is idle, then we first select among all eligible tasks (its predecessors have been completed) the task that has the shortest deadline for execution. If more than one task has the shortest deadline, we select the task that has the largest computation time. If still more than one task applies, we select an arbitrary one. If there are tasks for which time constraints can not be met, no feasible solution can be found. either the software partitioning should be redone (decompose a large tasks into a set of small) or high performance hardware should be used (increase the number of microprocessors and the speed of microprocessors). If the timing constraints of all tasks can be met, a feasible schedule has been found.

3.5 Implementation

After the distribution of the system over hardware and software, the tasks should be allocated and sheduled on parallel processors to achieve folowing goals:

1. to meet the behavioural requirements from the behavioural view;
2. to meet the timeliness constraints;
3. to meet precedence constraints between tasks;
4. to minimize communication cost between processors;
5. to balance utilization of each of the processors.

The main problems in the implementation from specification are how to manage the tasks on a multiprocessor and how do we transform the relations between functions of a functional specification into the relations between processors or into relations between tasks on a processor.

Task management encompasses several clearly distinguishable phases:

1. task assignment - the inital placement of tasks on processors;
2. task scheduling - local CPU scheduling of the tasks on a same processor;
3. task migration (load sharing) - dynamic reassignment of tasks to processors in response to changing loads on the processors.

Common heuristics result in suboptimumal solutions of task assignment, often useful in applications where an optimal solution is not obtainable. We suggest the application of the task assignment model of Chu and Lan [42]. In their algorithm, the load of each processor is built up from two components. *intermodule communication* (IMC) and *accumulative execution* time (AET) of each module. The task allocation function can be based on minimizing the load on the most heavily loaded processor ("bottleneck"). The parameter of precedence relation (PR) specifies the execution sequence of the modules. This algorithm for allocation involves the following steps:

1. compute IMC and AET;
2. compute the IMC index and the PR index;
3. combine modules with large IMC into groups to reduce total system load;
4. assign module groups to processors.

There has been much research into task scheduling on a single and multiprocessor system. Optimal schedules in most of these models require algorithms which are NP-hard. We use a suboptimal method which consists of the following steps:

1. determine the level l_i for each task of precedence graph (the level l_i of task i is defined to be the longest path from exit node to task i)
2. construct the priority list in the descending oeder of li and the number of immediately successive tasks.
3. execute list scheduling on the basis of the priority list.

Load sharing can be considered part of the larger distributed scheduling problem. Desirable properties of a load sharing strategy include:

- optimal overall system performance-total processing capacity maximized while retaining acceptable delays;
- fairness of service-uniformly acceptable performance provided to tasks regardless of the processor on which the task arrives;
- failure tolerance-robustness of performance maintained in the presence of partial failures in the system.

Load sharing must cause the tasks to be shared among the processors so that concurrent processing is enabled and bottlenecks are avoided whenever possible. A good load sharing algorithme will tend not to allow any server to be idle while there are tasks awaiting in the system. It will also not discriminate against a task based on the particular processor by which that task arrives.

We propose the application of the algorithm of Ni [44] for achieving Load Balancing. The algorithm is based on the identification of three distinguishable states for characterization of the processor load.

- a *light-load* state indicates that the processor can accept some migrant tasks:
- a *heavy-load* state indicates that it may be helpful to have some of the tasks to be migrated to other processors:
- a *normal-load* state indicates that no migration effort is needed for that processor.

The major bottleneck in implementing a communication protocol is again to meet the timing constraints of message-level and ensure the predictability requirements of real-time systems.

The relation "state variable" (see also section II) between functions is realized by memory. Main problem is to design entry and exit protocoles that satisfy the following properties:

- mutual exclusion;
- absence of deadlock;
- absence of unnecessary delay

Semaphores [5], [9], [48] are used for the implementation of critical sections, due to their simplicity in use.

However, task blocking time must be considered. Blocking occurs when a request is made for a resource which has mutual exclusion requirements and is already in use. The most common situation occurs when two tasks attempt to access shared data. To maintain consistency, the access must be serialized. If the higher priority task gains access first, then the priority order is manitained. However, if the highter priority task arrives after the lower priority task gains access to the shared data, the priority inversion taskes place. Priority inversion is said to occur when a higher priority task must wait for the processing of a lower priority task. If priority inversion is not controlled, it become impossible to determine whether tasks can meet their deadlines. To maintain a high degree of schedulablity of the system, communication protocol that would minimize the amount of the blocking are essential. We suggest to use A priority inheritance approach of R.Rajkumar [91] to solve this problem.

The relations "synchronization by event" and "transfer of information by ports " (see also section II) are realized by interrupt or boolean variables or semaphores or monitor or message passing [5], [9], [48].

When message passing is used. channels are typically the only objects process share. every variable is local to and accessible by only one process. This implies that variables are never subject to concurrent access. and therefore no special mechanism for mutual exclusion is required.

The programming can be done by the aid of the behavioural diagrams. The problem of translation of a behavioural diagram into parallel program involves: translation of states. translation of actions and translation of transitions expressed in this behavioural diagram. The methods of translation depend on the target parallel language to use. We do not discuss this issue here in detail.

4 A case study: CCSDS Telemetry

4.1 Introduction

CCSDS [34], [35] Advanced Orbiting Systems will include manned and man-tended space stations, unmanned space plateforms, free flying spacecrafts and advanced space transportation systems operating at a very wide range of user data rates in an environment of extensive on-board and ground computer networking.

Telemetry data accounts for a large fraction of the total volume of data that must be handled for typical space missions. Payload instruments generate data continuously, at rates which may vary although most instruments generate data at a rate of 1Mbits/s and only a relative few generate data at higher rates. Although data compression is occasionally incorporated to reduce communications requirements, most telemetry data are transmitted to ground "raw". Because of the high rates and volumes of data involved, their fast and efficient handling is essential. An important concept of packet telemetry is *segmentation*. Segmentation is a mechanism whereby the spacecraft data handling system breaks long packets into shorter pieces, with additional data for reconstructing the long packets.

In order to meet the requirements for on board telemetry systems, we developed a system which can generate the transfer frame telemetry that complies to CCSDS Recommendation. The system is built using a transputer network.

4.2 The environment of the system

There are several satellites in the environment, one having several data sources which produce data to be transmitted to the telemtry system. The timing constraint is 2 milliseconds.

We begin by identifying the objects in the environment, as illustrated in fig. 2. Decomposition is finished once several objects in the environment are obtained, decomposition is completed at the level of data sources. Second, the operations required for each data source, must be identified. (The operation of the data source is merely to produce data and to output this data.) Next, we specify the time constraints on the operations, here one millisecond. Continuing our object-oriented development, we might write the specification of the data source as:

```
Data source ij is
   do
     produce data
     when data ready
        output data
        Timing constraint of output data on system
        implementation is
           While data ready
              demand  to execute  and finish the generation of transfer frame
```

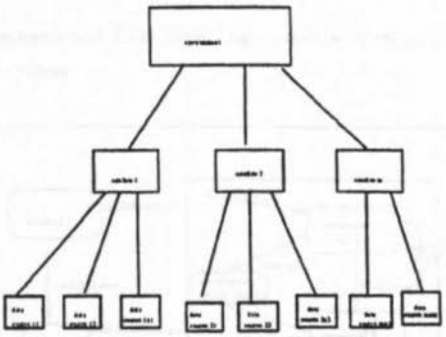


Figure 2: The environment of the system

```
                with  period =10 millesecond, deadline= one millesecond.
end  data source
```

4.3 Functional requirements of the system

The system must be able to furnish the transfer frame by packets such that its output complies to CCSDS Recommendations.

The primary functional requirements for the system, are as follows:

- creation of source packets. A source packet encapsulates a block of observation and auxiliary application data which is to be transmitted from a data source in space to a source analysis facility on the ground.
- multiplexing of the packets on a virtual channel. A single physical channel may be shared by different types of users by creating multiple apparently parallel "virtual" paths, a virtual channel, through the channel.
- Transfer trame generation. A transfer frame contains a header and trailer with space link protocol control information, and a fixed length data field with higher layer service data units.
- addition of CRC or R-S code [35], [35], and insertion of the synchronisation and serialisation messages for the generation of the Physical Channel Access Protocol Data Unit.

In figure 3, we specify the primary functions of the system using an extension of SADT. In this functional diagram, data ready, packet ready... are synchronisation events, frame length, packet length .... are state variables, source data, source packet... represent the transfer of information per port.

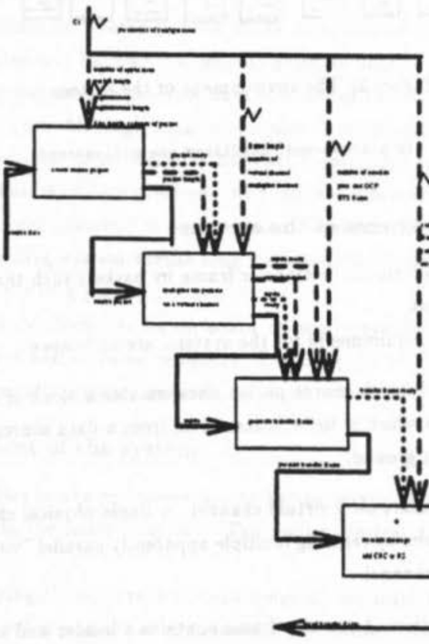


Figure 3: The functional requirements of the system

4.4 The behaviour of the system

The integration of statecharts and Real-Time Logic provides a suitable vehicle for the analysis of the behaviour of the system.

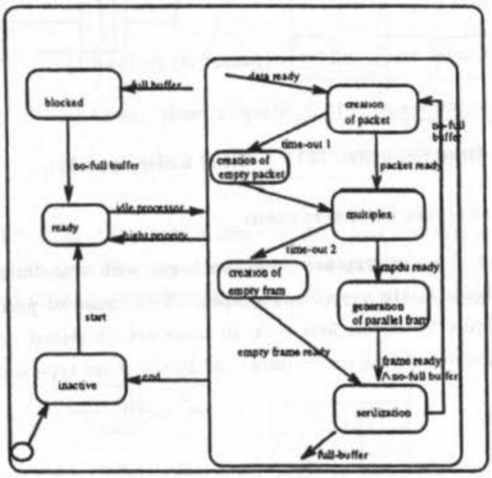


Figure 4: The behaviour of function generation of transfer frame

In fig. 4 a statechart is used for describing the behaviour of the system. At the first level, there are four states: *inactive*, *ready*, *blocked*, and *running*.
Initially, the function GTF (*G*eneration of *T*elemetry *F*rames) is *inactive*. When it receives the event *start*, it enters the state *ready*. In the state *ready*, the function GTF enters state *running* when the processor is idle. In that state it generates the transfer frame. If in state *ready*, the buffer is full, it enters the state *blocked*. When in state *blocked*, it enters the state *ready* when the buffer is not full. The function GTF completes its execution on receiving the event *end*, by entering the state *inactive*.

The state *running* is an abstraction for six sub-states: *creation of packets*, *creation of empty packets*, *multiplexing*, *generation of parallel frames*, *creation of empty frames* and *serilization*. Firstly, when source data is ready, GTF is in state *creation of packet*. When source packet is produced in an interval t1, the event *packet ready* occurs and the GTF enters the state *multiplexing*. If in that interval t1, no source packet is produced, a time out occurs and state *creation of empty packet* will be entered. Similarly, in state *multiplexing*, when a multiplexed data unit (mpdu) is produced in the interval t2, the event *mpdu* occurs and state *generation of parallel frames* is entered. If mpdu is not produced in that interval t2, a time out occurs and state *creation of empty frame* is entered. When the frame is ready and the buffer is not full, the function GTF enters the state *serialization*. In contrast, when the frame is 'buffer', GTF enters the state *blocked* to wait for the occurrence of the event *no-full buffer*.

Constraints upon events *time-out1*, *time-out2* and *no full buffer* are specified using Real Time Logic.

- time-out1= $\forall i$ @(packet ready, i+1) $\geq$ @(packet ready, i)+0.1)
- time-out2= $\forall i$ @(mpdu ready, i+1) $\geq$ @(mpdu ready, i)+0.1)
- no full buffer= $\forall i$ @(no full buffer, i+1) $\leq$ @(full buffer, i)+0.01)

4.5 Performance analysis of the system

For the construction of a performance view, we begin with transforming the functional diagram as shown in figure 3 into precedence graphs. Two types of precedence graphe are possible, as shown in figure 5. In the first type all tasks are combined into a single task in order to minimize the amount of task communication. In the other type four tasks are used in a pipeline architecture.

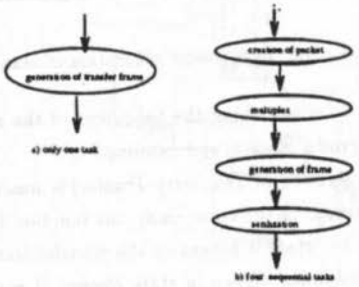
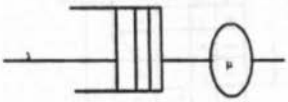


Figure 5: precedence graphe

For the evaluation of the two types of precedence graphs, we use queuing networks. For a transputer system (our target system), the system is modeled as the queue M/M/1. The results are shown in the Table 1. In this table, λ = data arrival rate (number/s), D= data arrival rate in Mbits (Mbits/s).

μ= mean service rate (10^4 bits). T= average response time (10^{-4} s)

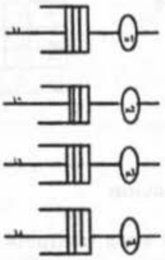
As shown in this table, the average response time is already 1 millisecond when $D = 9 Mbits$. Using a single transputer system does not lead to a solution that meets the performance requirement of the application.



λ	n	ρ	T
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]

Table 1: Execution on a Transputer

With a single task such that the data is allocated on a four transputer system. with statical configuration. the performance (the mean response times, and the utilization rate of the processor) is good, compared to the performance with a one transputer system.

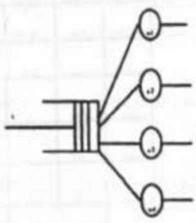


λ	n	ρ	T
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]

Table 2: Static allocation of data on four Transputers

When the precedence graph of the system is of the second type, each of the four tasks forming a pipeline. can be allocated on a single transputer. The results of performance simulation for this architecture are shown in table 4. The results show that with this choice of architecture the mean time response decreases, and the throughout increases, compared to the results presented in table 2. The results of table 2 and table 4 show little difference in performance between the two approaches. It must be noted, however. that the communication overheads are relatively small. However. when the tasks execute on different processors. transfer of information will cause overhead. Furthermore. having tasks waiting for communication with an otherwise idle processor, will degrade the performance of the system. Therefore, in the final implementation. we selected a single task on precedence graph of this application.

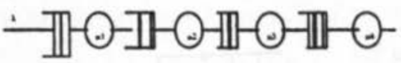
Based on the one task solution. the choice for an algorithm for dynamic allocation of data for the transputers, becomes interesting. The results in table 3, compared with the ones the ones in table 2 and table 4. show that the chosen dynamic allocation algorithm outperforms the static allocation algorithm using the same level of information. This phenomenon can be



λ	D	ρ	T
1000	[illegible]	[illegible]	[illegible]
4000	[illegible]	[illegible]	[illegible]
8000	[illegible]	[illegible]	[illegible]
16000	[illegible]	[illegible]	[illegible]
20000	[illegible]	[illegible]	[illegible]

Table 3: Dynamic allocation of data on four Transputers

explained by noting realizing that the dynamic allocation algorithm does not allow a processor to become idle as long as data is waiting in the system.



λ	D	ρ	T
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]

Table 4: Four Transputers on pipeline

4.6 The material view and the implementation

A lab-grade prototype of the system was constructed using transputers. In order to meet the reliability requirement, recall that the CCSDS onboard system runs in a very hostil environment, we use nine T800 transputers. The executive software is written in OCCAM [21], [32].

The architecture of the system is given in figure 6, there are nine T800 transputers, one T222 transputer and an interconnection network. The root transputer is included in a PC . and is used as central controller of transputer networ. The other T800s are connected by the interconnection network. The T222 is used to control the connection of the network.
In the implementation, we use dynamic allocation of data, which is consistent with the performance view. The remaining T800's are organized in a way that one of the T800 processors is the management processor (root), where the load information of all the processors is gathered. When one of the other processors becomes idle or nearly idle, the root processor transfers tasks to it.

A connection between the root transputer and another processor is established through a simple protocol in which a message is send to the T222. After receiving this message, the T222 takes the appropriate actions. establishing a connection within 17 micro seconds.
Estimating the processor load is obviously a problem, our (simplified) assumption has been that there are n data on each processor. The root transputer calculates its load by the following

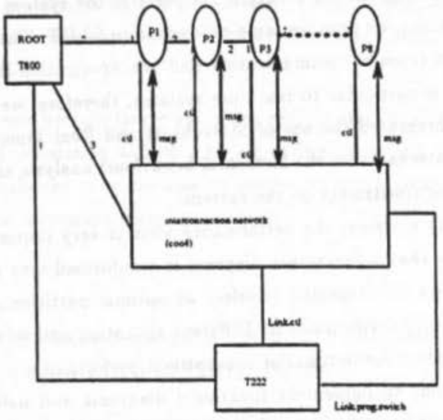


Figure 6: The architecture of hardware

formule:

$$load = (the\ length\ of\ data1/the\ packet\ length + ... + the\ length\ of\ data\ n/packet\ length)$$

The control algorithm on the root transputer is given in figure 7.

```
PROC manage( CHAN OF ANY data, proce, switch.in, swith.out, load)

      variables definition
 SEQ
   WHILE NOT end.execution
      ALT
        data ? dem    -inquire if there are arrival data
          SEQ
            load ? load_state
              to find the less loaded transputer
              IF
                processor i is underloaded
                  switch.out ! i  -output the number of transputer to create the connetion

                  clock ? time
                   ALT
                     switch ? ack  -waiting a ack message
                        proce ! dem     -output the data to processor i
                     clock ? AFTER time PLUS 17 microseconds
                        to process hardware fault and reconstruct the network
```

Figure 7: Controller algorithm

5 CONCLUSION

We have proposed a design methodology for modeling and development of multiprocessor real-time systems. The methodology is based on the identification of different views on the system to be build. and a decomposition of the system development accordingly. In our methodology, we have separated the environment and system clearly. environmentl modeling and analysis is one of the important elements in our methodology. In this environmental modeling. we proposed oriented object approaches.

For the support of the analysis of the functional aspects of the system (the functional view is our second view on systems), we propose some extensions to SADT. Three specific kinds of line are used to express three types of communication and synchronisation between functions.
The behavioural aspect is particular to real-time systems. therefore. we distinguish a separate behavioural view. We integrated the use of Statecharts and Real-Time Logic as a vehicle for behavioural analysis. Statecharts for the hierachial behaviourl analysis and Real Time Logic for the specification of timing constraints on the system.
In the design of real-time systems, the performance view is very important. It is during the construction of this view that a functional diagram is tranformed into a set of tasks and different partitioning schemes are evaluated to select an optimal partitioning with respect to the application and its operating environmment. Different allocation and scheduling algorithms are evaluated to select algorithms for optimal or sub-optimal performance.
We have proposed methods to implement functional diagrams and behavioural diagrams on multiprocessors, bridging the gap between the software specification and its implementation.
The example described in the previous section illustrates the use our methodology in the design of a real-time parallel system. The development of this application demonstrated that the proposed methodology is complete. It shows that the results of a thorough performance evaluation are useful for software partitioning and for the allocation of processes and data on a multiprocessor system. Furthermore, with performance evaluation the problem of garanteeing the deadlines of the various real-time tasks are addressed. Finally, the controlled utilization of a multiprocessor system improves the system's performance dramatically.
However, many research problems remain, we will briefly address some of these.

1. What (improved) modeling techniques can be used to model the user requirements and to formally validate these requirements;
2. What kinds of verification techniques are suitable for programs in this domain; Testing real-time software presents special problems, due to the practical impossibility to re-create particular state sequences that may occur as a reaction on external interrupts. Furthermore, even if such test were possible. the interpretation of the output results may be difficult. Analyticl techniques are therefore essential to verify the correctness of real-time software. Much research needs to be done in this aspect.
3. It is not possible to guarantee the reliabilty of a program by only validation and testing. If extremely reliable programs are required, some form of software fault-tolerance must be used. Much work remains to be done in this aspect.
4. For many real-time systems. it is not feasible neither for economic nor for safety reasons to stop or take off-line an entire system in order to change some parts of it. Hence, dynamic reconfiguration, needs to be supported. Much work remains to be done in this area.
5. Determining an optimal allocation and scheduling is known to be NP-hard and is hence

impractical. The problem is further complicated when, in addition to computation times and deadlines of tasks, resource requirements have to be accounted for. Much work remains also in this branch.

6. As computer systems become larger and more complicated, analytic modeling will become more difficult and simulation will play an increasing important role in performance evaluation. A key challenge is to be able to devise computationally efficient techniques to simulate models of increasing complex systems.

References

[1] U. de Carlini and U.Villano.*Transputers and parallel architectures: message-passing distributed systems.* ELLIS HORWOOD LIMITED, 1991.

[2] I.Granham and T.King, *The Transputer Handbook.* Prentice Hall, 1990.

[3] J.A. Stankovic, "Misconception about real-time computing: A serious problem for next generation systems," *IEEE Computer*, Vol.21, No.10, pp 10-19, October 1988.

[4] S.T.Allworth and R.N.Zobel. *Introduction to real-time software design.* Macmillan computer science series 1987.

[5] C.R.Andrews, *Concurrent programming.* The Benjamin /Cummings Publishing Company INC. 1991.

[6] F.Becceli and Z.Lui, "On the execution of parallel programs on multiprocessor systems -A queuing theory approach," *Journal of the association for computing machinery,* Vol.37, No.2, 1990.

[7] T.DeMarco, *Structured Analysis and System Specification.* Yourdon Press, ISBN 0-917072-07-3, 1978.

[8] D.Bustard, J.Elder and J.Welsh. *Concurrent program structures.* C.A.R. Hoare Series Editor. Prentice Hall, 1988.

[9] P.Chretienne, "The basic cyclic scheduling with deadlines," *Disc. APP. Math.,* VOL 30, PP 109-123, 1991.

[10] E.Gelenbe and G. Pujolle. *Introduction to networks of queues.* John Wiley, New York, 1986.

[11] H. Gomaa, "A software design method for real time systems," *Communications of ACM,* Vol. 27, No.9, 1984.

[12] K.Hwang, F.A. Brigges. *Computer architecture and parallel processing.* Mcgraw-Hill Book Company, 1984.

[13] INMOS, *Transputer development system 2.0.* INMOS, 1988.

[14] Estelle," a formal description technique based on an extended state transition model." *Draft International Standard ISO/TC97/SC21-DP9074*. September 1987.

[15] LOTOS."a formal description technique based on an the temporal ordering of observational behaviour," *Draft International Standard ISO/TC97/SC21-DP8807*. September 1987.

[16] L.Lamport. "What it means for a concurrent program to statisfy a specification: Why no one has specified priority," in *Conference record of the 12th annual A.C.M. Symposium on Principles of programming languages*, New Orleans. 1985.

[17] Z. Manna. P.Wolper. "Synthesis of communicating processes from temporal logic specifications," *ACM Transaction on programming languages and systems*, Vol.6. No.1 January 1984.

[18] A.Pnueli. "The temporal logic of programs," *18th I.E.E.E. Symposium on Foundations of computer Science*, pp.46-57, 1977.

[19] R. Milner. "A Calculus of Communicating Systems." In *Lecture Notes in Computer Science*. Vol. 92. Springer-Verlag. 1980.

[20] D.Pountain and D. May, *A tutorial introduction to OCCAM programming*. INMOS, March 1987.

[21] J.A. Stankovic and K.Ramamritham. "Hard real-time systems", *IEEE Computer Society*. Order Number 819. 1988.

[22] W.Reisig, *Petri Nets*, Springer-Verlag, Berlin Heidelberg New York Tokyo, 1982.

[23] INMOS. *IMS B008 User Guide*.1989.

[24] M.Jackson. *System development*. C.A.R HOARE series, Prentice-Hall 1983.

[25] Joint IECCA and MUF Committe on MASCOT (JIMCOM),*The Official Handbook of MASCOT*. Her Majesty's Stationery Office (June) 1987.

[26] P.T. Ward and S.J. Mellor, *Structured development for real-time systems*. Yourdon Computing series-YOURDON PRESS-Prentice-Hall.1985.

[27] D. Harel. "Statecharts, a visual formalism for complex systems." *Sciences of computer programming*. North Holland, Vol.8, 1987, P.231-274.

[28] HOOD Working Group. *HOOD Reference Manual*. draft B issue 3.0 (June 1989).

[29] G.Berry and G. Gonthier: "The synchronous language ESTEREL: Design, Semantics and Impementation", INRIA report 842, 1988.

[30] CCITT: "Functional specification and description language (SDL), Recommandations Z.101-Z.104." Vol.VI, Fasc. VI.7. Genova 1981.

[31] E. Hirsh. *Les Transputers: Application à la programmation concurrente.* Editions Eyrolles 1990.

[32] G. Pujolle and S.Fdida, *Modèles de systemes et de réseaux.* Editions Eyrolles. Paris. 1989.

[33] P.Richard. Y. Edward. Tsuchiya. "A task allocation model for distributed computing systems." *IEEE Transactions on Computers* vol. 31. No.1. pp 41-47. 1982.

[34] CCSDS. *Recommendation for Space Data System Standards. Packet Telemetry.* CCSDS Recommendation, Blue Book. Issue-2. January 1987 or later issue.

[35] CCSDS. *Advanced Orbiting Systems. Networks and Data links.* CCSDS 700.0-G-2. October 1989.

[36] I.G.L Technology, *SADT, Un langage pour communiquer.* Editions EYRolles, 1989.

[37] D.J. Haltley, I.A. Pirbhai, *Strategies for real-time systems.* Yourdon computing series-Yourdon Press Prentice Hall 1985.

[38] J.A. Stankovic, "A perspective on distributed computer systems," *IEEE Trans. Computer.* Vol c33, No. 12, pp 1102-1105, 1984.

[39] P.Heidelberger and S.S. Lavenberg, "Computer performance evaluation methodology," *IEEE Trans. Computer.* Vol c33, No.12, pp 1102-1105. 1984.

[40] R.A. Sahner and K.S. Trivedi, "Performance and reliability analysis using directed acyclic graphs." *IEEE Trans. Software Eng.* vol. SE-13, No.10. 1987.

[41] D.L. Eager, E.D. Lazowska, and J. Zahorjan, "Adaptive loading sharing in homogeneous distributed systems," *IEEE Trans. Software.* vol SE 12. No. 5, 1987.

[42] W.W. Chu and L.M-T. Lan, "Task allocation and precedence relations for distributed real-time systems," *IEEE Trans. Computer.* Vol. C36 No. 6. 1987.

[43] J.P. Huang, "Modeling of software partition for distributed real-time applications," *IEEE Trans. Software.* vol SE 11. No. 10, 1985.

[44] L.M.Ni. C.W.Xu, and T.B. Gendreau, " A distributed drafting algorithm for load balancing." *IEEE Trans. Software.* vol SE 11, No. 10, 1985.

[45] D.W. Leinbaugh and M.R.Yamini. "Guaranteed response Times in a Distributed Hard - Real-Time Environment." *IEEE Trans. Software.* vol SE 12, No. 12, 1986.

[46] S.S.Yau and J.J.P.Tsai. "A survey of Software Design Techniques," *IEEE Trans. Software.* vol SE 12. No. 6. 1986.

[47] A. Dinning. "A survey of synchronization methods for parallel computers. Computer." *IEEE Computer Society* July 1989.

[48] A. Thomasion and P.F. Bay. "Analytic queueing network models for parallel processing of task systems," *IEEE Trans. Computer.* Vol. C-35. No.12. 1986.

[49] C.C. Roman. "A Taxonomy of current issues in requirements engineering, Computer," *IEEE Computer Society*, April. 1985.

[50] D.T.Ross, "Applications and Extensions of SADT." *IEEE Computer Society.* April. 1985.

[51] M. Alford, "SREM at the age of eight: the distributed computing design system." *IEEE Computer Society.* April. 1985.

[52] Sol M. Shatz and J.P. Wang, "Introduction to distributed-software Engineering," *IEEE Computer Society.*, October, 1987.

[53] Hoare. C.A.R., "Communicating Sequential Processes." *Comm. ACM.* Aug. 1978, pp. 666-677.

[54] J.A. Stankovic, K. Ramamritham, and S.Cheng, " Evaluation of a flexible Scheduling algorithm for distributed Hard Real-time Systems, IEEE Trans. Computer." Vol.C-34, No.12, 1985.

[55] W. Zhao, K. Ramamritham, and J.A. Stankovic, "Scheduling Tasks with Resource Requirements in Hard Real-time Systems," *IEEE Trans. Software.* vol SE 13. No. 5, 1987.

[56] M. Molloy, "Discrete Time Stochastic Petri Nets," *IEEE Trans. Software Eng.* vol SE 11, No. 4, 1985.

[57] Y.T. Wang and R.J.T. Morris, "Load sharing in distributed systems," *IEEE Trans. Computer.* Vol. c-34, No.3, 1985.

[58] B. Auernherimer and R. A. Kemmerer, " RT-ASLAN: A specification Langage for Real-time Systems," *IEEE Trans. Software Eng.* Vol. Se-12. No. 9, 1986.

[59] F. Jahanian and A.K. Mok, "Safety Analysis of Timing properties in Real-time systems," *IEEE Trans. Software Eng.* Vol. Se-12. No. 9, 1986.

[60] R.G. Fichman and C.F. Kemerer, "Object-oriented and conventional analysis and design methodologies," *Computer, IEEE Computer Society.* October 1992.

[61] P. T. Ward, "The Transformation Schema: An Extension of the data flow diagram to represent control and timing," *IEEE Trans. Software Eng.* Vol. Se-12. No. 2, 1986.

[62] G.Booch. "Object-oriented Development." *IEEE Trans. Software Eng.* Vol. Se-12. No. 2, 1986.

[63] J.R. Cameron, "An overview of JSD," *IEEE Trans. Software Eng.* Vol. Se-12. No. 2, 1986.

[64] L.Zhang, M.Galindo, D. Marquie and Y. Raynaud. "Methodology of real-time system design using multiprocessors". *Microprocessors and Microsystems.* Vol 17. No 14, may 1993.

[65] W. A. Halang, "Real-time systems: Another Perspective", *The Journal of Systems and Software.* April 1992. pages 101-108.

[66] A.C.Shaw, "Towards a timing semantics for programming languages". *Foundations of real-time computing: Formal specifactions and methods.* Kluwer academic publishers. 1991.

[67] B.Hoogeboom and W.A. Halang, "The concept of time in the specification of real-time systems", in [68].

[68] K.M.Kavi. *Real-time system: abstractions,languages, and design methodologies.* IEEE Computer Society Press, 1992.

[69] K.M.Kavi and S.M.Yang "Real-time systems design methodologies: An introduction and a Survey," *The Journal of Systems and Software*, April 1992, pages 85-99.

[70] A.D.Stoyenko, V.C. Hamacher, and R.C.Holt "Analyzing hard real-time programs for guaranteed schedulability" *IEEE Trans. Software Eng.* Vol. Se-17. No. 8. August 1991, pages 737-750.

[71] J.S.Ostroff "Formal methods for the specification and design of real-time safety critical systems" *The Journal of Systems and Software*, April 1992, pages 33-60.

[72] W.A.Halang and A.D. Stoyenko. *constructing predictable real time systems*, Kluwer Academic Publishers, 1991.

[73] A.M.V. Tilborg and C.M.Koob, *Foundations of real-time computing: Formal specifactions and methods.* Kluwer Academic publishers, 1991.

[74] A.M.V. Tilborg and C.M.Koob.*Fundations of real-time computing: Scheduling and Resource Management.* Kluwer Academic publishers, 1991.

[75] H.Gomaa*Software design methods for concurrent and real-time systems.* Addison-Wesley Publishing Company, 1993.

[76] J.Xu and D.L. Parnas, "On satisfying timing constraints in hard real time systems", *Proceeding of the ACM SIGSOFT'91 Conference on Software for Critical Systems,* New Orleans. Louisiana. December 4-6,1991.

[77] S.T.Levi and A.K.Agrawala.*Real-time system design* McGraw-Hill international Editions. 1990.

[78] M.Schiebe and S.Pferrer. *Real-time systems engineering and applications.* Kluwer Academic Publishers. 1992.

[79] C.Y.Park and A.C Shaw, "Experiments with a program timing tool based on source level timing schema", *Readings in Real-time systems. Edited by Y.H.Lee and C.M.Krishna*, IEEE Computer Society Press, 1993.

[80] Y.H.Lee and C.M.Krishna. *Reading in Real-time systems*. IEEE Computer Society Press. 1993.

[81] C.Y.Park, "Predicting program execution times by analyzing static and dynamic program paths" *Real-Time Systems*, 5, 1993, pages 31-42.

[82] E.V.Sorensen, J.Nordahl and N.H.Hansen, "From CSP models to Markov models" *IEEE Trans. Software Eng.* Vol. 19. No. 6, June 1993, pages 554-570.

[83] C. U. Smith and L.G. Williams. "Software performance Engineering, A case study including performance comparison with design alternatives", *IEEE Trans. Software Eng.* Vol. 19. No. 7, July 1993.

[84] D. A. Menasce, "A Methodology for performance evaluation of parallel applications on multiprocessors" *Journal of parallel and distributed computing*, 14, 1992 pages 1-14.

[85] A.Sowmya, "A statecharts-based specification and verfication of real-time job scheduling systems", Real-time programming, IFAC Workshop, Bruges, Belgium, 23-26 June 1992.

[86] S.Prehn and W.J.Toetenel, "VDM'91: Formal software development methods", LNCS 551, Springer-Verlag, 1991.

[87] L.Zhang, "Une méthodologie de conception des applications temps reél destinées a être implantées sur des machines cibles multiprocesseurs", Ph.D disseration, 1993.

[88] N.Plat, J.van Katwijk, and W.J. Toetenel," Applications and benefits of formal methods in software development." *IEE-BCS Software Engineering Journal*, 7(5):335-346, 1992.

[89] G.Booch, *Software engineering with ADA*. The Benjamin/Cummings Publishing Company INC, 1983.

[90] Liu.C.L. and Layland J.W, "Scheduling algorithms for multiprogramming in a hard real-time environment.", JACM 20 (1):46-61, 1973.

[91] R. Rajkumar. *Synchronization in real-time systems : a priority inheritance* Kluwer academic publishers, 1991.