4-1997

Predicting Timing Behavior in Architectural Design Exploration of Real-Time Embedded Systems

Rajeshkumar Sambandam
Western Michigan University

PREDICTING TIMING BEHAVIOR IN ARCHITECTURAL DESIGN EXPLORATION OF REAL-TIME EMBEDDED SYSTEMS

by

Rajeshkumar Sambandam

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science in Engineering
Department of Electrical and Computer Engineering

Western Michigan University
Kalamazoo, Michigan
April 1997

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my major thesis advisor, Dr. Sharon Hu, for her unstinting support and guidance throughout this work. I would like to express sincere thanks to my committee members Dr. Garrison Greenwood and Dr. Frank Severance for their strong encouragement and reviews on this work. I am grateful to Dr. Severance for extending his laboratory facilities for this thesis. On a personal note, I wish to thank all my friends for their timely help, cooperation and understanding in the completion of this thesis.

Rajeshkumar Sambandam

PREDICTING TIMING BEHAVIOR IN ARCHITECTURAL DESIGN EXPLORATION OF REAL-TIME EMBEDDED SYSTEMS

Rajeshkumar Sambandam, M.S.E.

Western Michigan University, 1997

This thesis aims at developing efficient analytical techniques to perform trade-off studies in the architectural design exploration phase of real-time embedded systems. Real-time embedded systems (RTES) have stringent timing requirements. In the process of designing such a system, a key issue is to determine if a system is able to meet all the timing requirements imposed on it, though it may prove attractive in terms of cost and performance. Traditional methods of using event-driven simulation for modern RTES are very time consuming and are not guaranteed to prove feasibility. Specifically, we identify a *flexibility* metric to compare various options in the design exploration task. This metric is obtained based on certain schedulability bounds. The famous rate monotonic scheduling theory was studied in detail. The new metric was evaluated by comparing its results with the results of simulation for a real-world system as well as systems composed of randomly generated tasks. Experimental results show that our approach can be effectively used in the design evaluation stage of many real-time embedded systems.

TABLE OF CONTENTS

Table of Contents–Continued

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Embedded systems encompass a variety of hardware and software components which perform specific functions in host systems. Examples of such systems include CNC (computer numerically controlled) machines, aircraft avionics, defense systems and most automated systems in modern factories. Many embedded systems in real life, must respond to external events under certain timing constraints. Failure to respond to certain events may either seriously degrade system performance or even result in a catastrophe. Such systems are referred to as real-time embedded systems (RTES) and can be found in many applications, such as engine and transmission control of automobiles, navigation and landing control of aircraft, and communication networks. In designing real-time embedded systems, in addition to the usual design criteria for embedded systems, such as expandability, reliability, maintainability and cost-effectiveness, a key issue is to guarantee that the system can provide timely services.

For a given RTES, one can identify the various functions which have to be implemented using computers without much difficulty. A major challenge lies in deciding which tasks should be implemented in dedicated

hardware circuits and which should be in software so that optimal cost, performance and low power dissipation can be achieved. In order to compete effectively, RTES must also meet various design constraints with short time to market and at low cost. To efficiently explore various system architectural options, a number of researchers have proposed to use the system-level or configuration-level modeling ([1,3,4,8,12,22,24]). At this level, hardware is modeled as resources with no detailed functionality and software is modeled as tasks utilizing the resources.

One of the major issues to be considered during the system level architectural design phase of RTES, is the *feasibility* of the system architecture. This refers to determining if the processor is capable of meeting the timing constraints of all software tasks currently assigned to it. Various real time analysis techniques have been devised to check if the processor is able to finish each software task before its deadline. Another related issue is the system's *flexibility*. This refers to the processing power yet available from the processor to process any possible future tasks or tolerate perturbation of current tasks. Determining the flexibility of a RTES is an important issue because in most cases, the system specifications may not be completely available. Even if it is fully specified, it is often the case that the specification will be altered from the time of the first formulation of the system requirements to the time of the final implementation of the

system. Furthermore, information on software tasks (e.g., execution time and memory requirements) and hardware components (e.g., power consumption and chip area) is quite often based on some kind of estimates. Thus, it is desirable to have system designs which are more flexible and can tolerate wide ranges of perturbation. Nonetheless, if there is a flexibility metric, it can be used for trade-off studies during the architectural design exploration.

One technique to determine the timing requirements of a RTES is by means of event-driven simulation. However, this technique is not applicable in practice since an extensive simulation is very time consuming and may not necessarily prove feasibility. Also, slight changes in the specifications would mean expensive new simulation of the whole system. The complexity of modern real-time systems makes it impossible to simulate all possible situations.

Some researchers have proposed analytical and algorithmic techniques [14, 15, 17] to predict the timing behavior of RTES. However all these methods intend to provide a *yes* or *no* answer to the question of whether a system meets its timing specification. Given the *binary* nature of such real time analysis approaches, one may have difficulty to use them for measuring the flexibility of a system architecture. This thesis concentrates on arriving at a flexibility metric with respect to timing behavior in

real time embedded systems. We analyze certain existing real time analysis approaches [8, 15, 17] to arrive at a flexibility metric and identify the applicability of these metrics in evaluating initial system designs by verifying through simulation.

The remainder of this thesis is organized as follows. In Chapter II, we review some of the related work. This chapter also introduces some scheduling techniques and existing real-time analysis approaches in brief. Chapter III presents our experimental set-up in comparing various flexibility metrics. In Chapter IV, we compare several potential metrics from the real-time analysis field and identify some applicable metrics and illustrate their effectiveness. Conclusion and further work are discussed in Chapter V.

CHAPTER II

RELATED RESEARCH

Timing analysis for real-time systems are critical during the architectural design exploration stage. The goal of timing analysis is to check if all the software tasks assigned to a processor meet their deadlines. At the system architectural level, each software task (τ_i) is associated with the following parameters: computation time (c_i), deadline (d_i), period (p_i) and activation (a_i). In the following paragraphs, we will introduce some terminologies commonly used in the real-time analysis field and describe some scheduling techniques and existing real time analysis approaches.

Relevant Terminologies

1. *Task*: A task is any software routine that will be executed by a processor.

2. *Period*: The period of a task is the inter-arrival time between any two successive requests of the task. This is also the reciprocal of the frequency.

3. *Computation time*: This is the time the processor takes to *execute* a request from a task. If n is the number of instructions of a task and P is

the MIPS rate of the processor, then the computation time of this task would be nP μsecs.

4. *Deadline*: The deadline of a task is the *latest* time by which a given request can be completed by the processor. Each task instance also has a deadline which will vary from request to request. For e.g., if the period of a task is 20 and its deadline is 17, then the deadline of the 5th instance of this task would be (5-1)x20 + 17 = 97, which is measured from time zero. Note that the first instance of any task occurs at time zero.

5. *Activation*: This is the time for which a task is held after it is instantiated and before it makes a request for execution to the processor.

6. *Release time*: This is the time at which a given instant of a task makes a request to the processor. For e.g., if the period of a task is 10 and its activation is 2, then the release time of the 10th instant of this task would be (10-1)x10 + 2 = 92.

7. *Response time*: The response time of an instance of a task is the time interval between its release time and the completion time of the request of this instance of the task by the processor.

8. *Task set*: This is the set of all tasks assigned to a processor. Each task in a task set has their own period, computation time, deadline and activation.

9. *Feasible system*: A real time embedded system is said to be feasible if all tasks in the task set meet their deadline.

10. *Critical instant*: The critical instant for a task is defined to be an instant at which a request for that task has the largest response time.

11. *Worst case phasing*: The worst case phasing for a task set occurs when the activation of all tasks in the set is zero.

12. *Worst case situation*: Let S be the set of tasks and define $S'(t) \supseteq S$. $S'(t)$ contains the tasks that must be executing at time t if tasks are to meet their deadlines. Then, the worst case situation is given by $\max_t |S'(t)|$.

Scheduling Techniques

Task Allocation Schemes

An embedded system may contain more than one processor and hence the execution of a task can be allocated to any processor in the system. Two different task allocation schemes exist: static and dynamic [5,6,19,20,21]. Static task allocation is applied when the execution properties of all processes are known in advance. On the other hand, dynamic task allocation is used whenever the task execution properties are not completely known in advance. In our following discussion, we assume that

a fixed allocation is given and concentrates on the scheduling analysis of a single processor system.

Scheduling Strategies

Given a processor and a task set, a scheduling strategy must be developed that describes the rule of assigning priorities to tasks and whether the tasks are pre-emptable. A number of researchers have studied the scheduling analysis of real-time systems (e.g., [2,7,15,16,18,23,26]). The pre-emptive scheme, under which the execution of a lower priority task can be interrupted by a higher priority task, has become the most widely accepted scheme in RTES. Thus, we will assume that the systems we study make use of the pre-emptive scheme.

Priority Assignment Rules

The rules of assigning priorities for tasks in a given task set can be classified into two categories: dynamic (changing priority) and static (fixed priority).

Static Scheduling Approach. In static scheduling approach, the priorities of the tasks for every request for execution to the processor remain the same at all times. One of the dominant static scheduling approaches is

the rate monotonic (RM) approach [14,15,17]. The RM algorithm assumes that task requests are made periodically. It assigns priorities to tasks according to the frequencies of their execution requests. The task with the highest request frequency i.e., the smallest request period, gets the highest priority.

Dynamic Scheduling Approach. In this approach, the priority of a task may change from request to request. A prevalent dynamic scheduling approach is the deadline driven algorithm. This algorithm assigns the highest priority to a task if the deadline of its current request is the nearest and the lowest priority if the deadline of its current request is the furthest. Liu and Layland in [17] have proved that the deadline driven algorithm to be an optimal dynamic algorithm.

In RTES, static scheduling is more widely used than the dynamic approach and hence our main focus in this thesis is on RM scheduling approach. Also, the RM scheduling algorithm has been proven to be the optimal static scheduling algorithm [17]. Furthermore, it has the following advantages [10,15]: First, it can be used to ensure that the timing requirements of the critical tasks are met in case of a transient overload in the system. Second, aperiodic tasks can be easily handled while still meeting the deadlines of periodic tasks. Third, this technique can be eas-

ily used where imprecise computation is required. Lastly, it is easy to implement in processors, in I/O controllers and in communication media.

RM Scheduling: Example

Figure 1 shows the execution sequence (in Gantt Chart form) based on RM scheduling for a three task system given in Table 1. In this figure, $w(i,j)$ represents the response time for the j^{th} instance of task τ_i. Notice

Table 1

A Sample System to Illustrate RM Scheduling

i	p_i	c_i	a_i
1	100	20	5
2	150	40	7
3	300	50	5

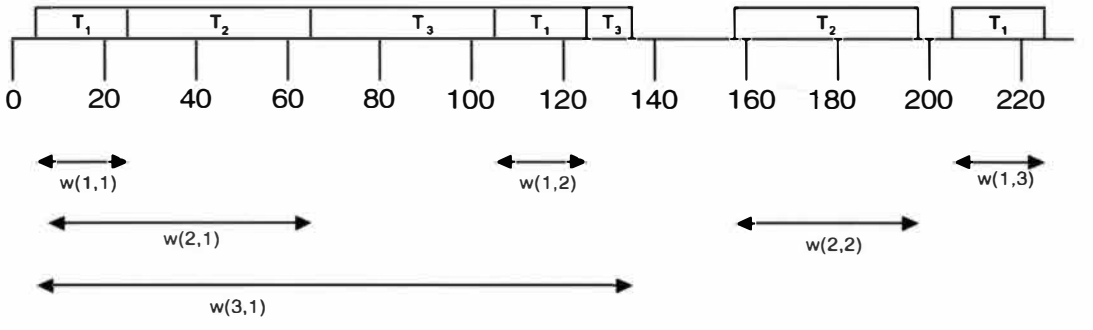

Figure 1. Gantt Chart Based on RM Scheduling for the System Given in Table 1.

that this time-line is drawn assuming the pre-emptive scheme. The 7,157,307,... and 5,305,605,... respectively. Consider the first instance of lease times for various instants for task τ_1 ,τ_2 , τ_3 are 5,105,205,..; task τ_3. Its release time is 5. However it starts execution only at time 65.when the execution of the first instant of τ_2 is completed. Pre-emption of task τ_3 occurs at time 105 when the second instance of task τ_1 arrives.

Existing Real-Time Analysis Approaches

Given a task allocation, the pre-emptive scheme and a fixed-priority scheduling algorithm, several researchers have proposed feasibility decision algorithms based on different techniques such as event driven simulation, average processor utilization [15,17] and processor response times [23]. In the following paragraphs, we will briefly discuss these methods and identify their merits and demerits.

Simulation Based Approach

A straightforward method to determine if the timing requirements of all tasks are met would be to simulate the system execution for the given schedule. This requires that a time interval within which to construct the schedule be defined. The interval should be as short as possible to reduce the simulation time. Yet it needs to be long enough so that the

schedulability in this interval guarantees the schedulability over the entire time span. In [12], Lawler and Martel proved the existence of such an interval, $[a_{max}, a_{max} + 2P]$, where P is the effective period of the task system, and is equal to the least common multiple of all task periods, and a_{max} is the maximum among all task activation times. Unfortunately, this interval can become very large and computationally inhibiting when task periods are relatively prime. Thus, this exhaustive means of verifying the timing performance is impractical for large complex systems. It has been shown that predicting the timing behavior of a task set allocated to a single processor is an *NP*-complete problem [16].

Average Processor Utilization Based Approach

One of the prevalent approaches in determining feasibility is rate monotonic analysis (RMA) [14,15,17]. The RMA technique uses the average processor utilization defined in (1) in estimating the timing behavior of RTES.

$$U = \sum_{i=1}^{n} \frac{c_i}{p_i} \qquad (1)$$

Liu and Layland, in [17] have demonstrated that, whenever the worst-case phasing exists and when the period of each task equals its deadline, the RM algorithm will guarantee a feasible schedule provided that the

task utilization factor, U, is bounded by

$$U=\sum_{i=1}^{n}\frac{c_i}{p_i}\leq n\left(2^{\frac{1}{n}}-1\right) \qquad (2)$$

However, in many systems, the RM algorithm can often feasibly schedule task sets having total utilization higher than the bound given above. This has been shown by Lehoczky, Sha and Ding in [15]. In fact, it is not uncommon to have systems even with an average utilization around 0.9 and that are schedulable by RM algorithm. While Lehoczky 's technique [15] provides a significant improvement from the Liu- Layland's bound in determining the feasibility, their method cannot be directly used for obtaining the flexibility of a system architecture at the design exploration stage since it is still based on worst case analysis[1].

Response Time Based Approach

Another widely used method to predict the feasibility of RTES is based on comparing the deadline of each task with the worst case response time of the processor. Lehoczky in [13] proposed the "*level-i*" busy period concept to determine the longest response time for the request of a task initiated at a critical instant. This technique was also used by Yen and Wolf [23] to determine the worst case response time through a fixed

[1] In the last part of this chapter, we give more details on this.

point iteration technique. They also considered the case where tasks are not initiated at a critical instant. They model this by dummy tasks with data dependencies. Their approach can be computationally expensive considering that many system configurations may need to be investigated at the architectural design stage. Joseph and Pandya [10] have suggested a interval arithmetic method to obtain the response times. However, their method is again, based on the worst case phasing.

Need for a Flexibility Metric

Using the above techniques based on worst case phasing has been widely accepted as an effective means to answer whether a given system is feasible. In reality, we often confront with system design which do not have worst case phasing. Depending on particular task compositions, the situation corresponding to worst case phasing may never occur for systems which have activation. Thus, if these techniques when used to evaluate design options for systems which have activation may lead to a "over-specification" of the timing requirements. Therefore, these "pessimistic approaches" when directly applied in the system architectural design stage, some potentially valid system configurations may be thrown away.

We later show in our thesis, how other approaches based on lower bound on processor utilization and certain schedulability bounds, feasibility factor etc. can be effectively used to overcome the shortcomings of pessimistic approaches. We also provide an analysis on these metrics in terms of its effectiveness in reflecting the change in timing requirements due to the change in timing parameters of the system. These results gives an estimation for the system designer, "how close" the system configurations meet the timing requirements. They can be later used to study trade-offs on system design specifically involving timing details.

CHAPTER III

EXPERIMENTAL SET-UP FOR COMPARING POTENTIAL METRICS

The goal of this thesis is to identify a metric that can be used to measure the flexibility of a real-time embedded system during the architectural design exploration phase. This requires a test environment where we can study and compare certain potential metrics. We used a real-world embedded system and systems composed of randomly generated tasks[2] (RGT) instead of hand-crafted toy systems in our study. A commercial simulation tool–SES/Workbench[3] was used as an event-driven simulator to compare the results of various analytical methods. In this chapter, we will first describe our real-world example and the generation of RGT systems. Later, we discuss the SES model used for the simulation.

The Engine Control System

We use an example presented by Hu, *et. al.* [9] to study the timing requirements of a typical embedded computer system. The example involves the design of a real time embedded system implementing a subset

[2] Hereafter, we will refer to these systems as RGT systems.
[3] Product of Scientific and Engineering Software, Inc., Austin, TX.

volves the design of a real time embedded system implementing a subset of an engine control module. Since our concern is only on the timing details, we directly use the workload specifications given in [9] for our study. This is shown in Table 2. In this table, p is the minimum time interval between engine events and μs denotes micro-seconds.

The instruction count of each function represents the worst case (dynamic[4]) length of the primary section of the task. The primary section is the portion of the function that is executed under normal operating conditions and excludes other parts of the function containing the code for handling extreme situations which are not time critical. The actual number of instructions is estimated based on previous versions of engine control programs and current control algorithm complexities. Certainly, these estimates depend on the particular processor on which the function is running. The numbers shown in Table 2 are based on a generic RISC architecture. Each function has a deadline and activation which may be expressed either in p or units of time. The numbers in parenthesis show the μs equivalent (assuming an engine speed of 6000 rpm) of the values expressed in engine events. The frequency of a function indicates how many times in one engine revolution, the function is executed. The numbers in

[4] If the length of the software routine that implements the function is dependent on a factor that could be decided only at run-time, then we consider the worst-case value of the factor.

parenthesis in this column show the period (the reciprocal of frequency) in μs.

Table 2

Workload Specifications of an Engine Control Module

Task	**Instr. Count**	**Deadline**	**Frequency-1/rev (Period)**	**Activation**
DigitalFilter1 (DF1)	64	46 μs	96 (104.17)	0 (0)
DigitalFilter2 (DF2)	32	1 *p* (10000)	1 (10000)	95 *p* (9895.93)
DecodeSPUB (DSB)	30	83 μs	48 (208.33)	0 (0)
DecodeSPUA (DSA)	30	55 μs	48 (208.33)	83 μs
ReadCAM (RC)	30	4 *p* (416.67)	1 (10000)	0 (0)
ServiceRoutine (SR)	20	2 *p* (208.33)	24 (416.67)	0 (0)
FuelCalc (FC)	480	500 μs	4 (2500)	8 *p* (833.33)
SparkCalc (SC)	100	8 *p* (2500)	4 (2500)	16 *p* (1666.67)
ReadMAP (RM)	40	3 *p* (312.5)	24 (416.67)	0 (0)

In order to construct a reasonable set of data points, we assume that there are various architectural designs that contain different combinations of the nine tasks implemented either as hardware or software. For instance, a system configuration could be such that DF1 and DF2 are implemented as software running on a processor and the remaining tasks are implemented as dedicated hardware. Therefore, the processor needs to execute only DF1 and DF2. Likewise, we can construct up to 2^9 system configurations with different task combinations to be executed by a processor. The processor is from a pool of host processors listed in Table 3. The list of host processors is obtained by taking advantage of the minimum and maximum operating clock frequencies for a given processor. We

Table 3

MIPS Rate of Various Host Processors on Which Software Tasks Were Executed

Processor	MC1	MC2	MC3	MC4	MC5
MIPS	1.30	1.35	1.43	1.50	1.70
Processor	MC6	MC7	MC8	MC9	MC10
MIPS	2.00	2.10	2.20	2.30	2.50

assume that we will be able to arrive at the various MIPS rate identified in Table 3 by varying the clock frequency of the processor. It is essential

for us to have a varying MIPS rate in our study because it determines the computation time of each task which is an important constituent in the overall timing behavior of the system.

RGT Systems

A timing analysis based on the example embedded computer system such as the engine control system described above may capture many characteristics typically found in real time systems. However, one cannot generalize the results obtained from the example system since all systems found in real-world may not behave in the same fashion in terms of timing requirements. For instance, in the above system, the period and deadline of some of the tasks are integral multiples of each other. Clearly, the timing behavior could be much different if the numbers are relatively prime. Hence, the results we obtain for this system could be a biased one. In order to ensure a fair comparison, it is imperative to perform timing studies on RGT systems.

To compose an RGT system, we need to generate the parameters such as period (p_i), deadline (d_i) , activation (a_i) and computation time (c_i) for each task (τ_i) in the system. We chose to use a congruential pseudo random number generator which returns numbers distributed uniformly, to generate random numbers over a given range. The ranges for period

and computation time in our RGT systems were between [10,8500] and [2,950] respectively. (Unlike the engine control system, we use the computation time of a task instead of instruction count and hence the processor MIPS rate is no longer a factor in the timing behavior of RGT systems). The lower and upper limits of period and computation time were chosen so as to obtain reasonably good number of uniformly distributed values for upper bound, lower bound and feasibility factor[5] of systems. Another limitation on the upper limit of periods is the fact that having a higher value for periods would lead to unnecessarily long simulation time. Clearly, the deadlines for any *meaningful* system should be greater than their corresponding sum of computation time and activation. Also, since we do not consider systems with deadline greater than period, the range of the deadline is determined for each task individually. That is, for task τ_i, its deadline is a random number within $[c_i, p_i]$. Fixing the above ranges for period, computation time and deadline lead us to determine the valid limits for the activation time as $[0, d_i\text{-}c_i]$. However, having such a wide range for the activation time is not quite practical and often results in generating unacceptable[6] systems. Therefore, we reduce the range to [0,

[5] The meaning of these terms will become clear in the next chapter.
[6] A system is unacceptable if the value of a metric (under consideration) is not within the range of our study. Legal values of a chosen metric are explained in the next chapter.

$r(d_i - c_i)$] where r is any random number between [0.1,0.9].

In order to do a fair comparison between various metrics that we will be discussing later, it is necessary to generate meaningful systems which will be fairly uniformly distributed over the legal values of the metrics. The procedure given in Figure 2 illustrate our approach for generating RGT systems. It is evident from this figure that not all numbers returned by the random number generator are suitable to be included in a system. Therefore, at every stage in the generation of an RGT system, we make certain validation tests. For instance, the utilization test ensures that we do not include systems which have average processor utilization greater than 1. Such systems are obviously, infeasible. The *balance_check* routine performs a test of uniformity among the systems generated. The inclusion of this procedure ensures that systems are generated fairly in equal numbers in all ranges of the metrics we studied.

In the analysis based on RGT systems, each task set contains nine tasks, which is the same as in the example system described in the last section. The number of tasks in a system is not a matter of critical concern in our study since we are mainly interested in the analysis of relative timing behavior of potential systems at the architectural design stage. However, the number of tasks would become an important factor when we compare the computational needs of each potential metric. We will discuss

more on this aspect when comparing the metrics.

```
procedure: RGT System Generator

    begin{
        while number of tasks generated ≤ number of tasks required){
            generate period within [10, 8500]
            generate computation time within [2, 950]
        }
        check for a legal value of processor utilization;
        if(acceptable processor utilization){
            for(each task generated above){
                Generate deadline within [c_i, p_i]
                Generate r within [0.1, 0.9]
                Generate activation within [0, r(d_i-c_i)]
            }
            compute the value of the metric under investigation
            if(acceptable value of the metric){
                procedure: balance_check;
                if(balance_check==FALSE){
                    discard this task set;
                    begin new system;
                }
            }
            else {
                discard this task set;
                begin new system;
            }
        }
        else begin new system;
    }
end procedure
```

Figure 2. Procedure to Generate RGT Systems.

System Modeling and Assumptions

In order to measure the effectiveness of the various analytical methods that we studied, an SES/Workbench model was developed to perform simulation of the engine control system and RGT systems described above. The results of the simulation for every possible task combination derived from the engine control system and RGT systems will be compared against the values of various metrics to analyze its effectiveness in estimating the flexibility of a system architecture.

SES/Workbench is a simulation tool that is used to evaluate the correctness and performance of a system design. Event-based simulation forms the basis of this tool. The main module page of the SES model for the engine control system is given in Figure 3. The various requests from the tasks in the SES model are processed as *transactions*. Transactions are invisible entities that flow from node to node along the arcs. Every transaction along the arcs on the graph, either picks up data from the nodes or the nodes process the data carried by the transactions in order to divert them through the right path. For instance, in the engine control system, an architecture could have DF1, DF2 and SC implemented as software and the remaining tasks as hardware. This configuration would be modeled in the SES as follows: transactions that get generated from

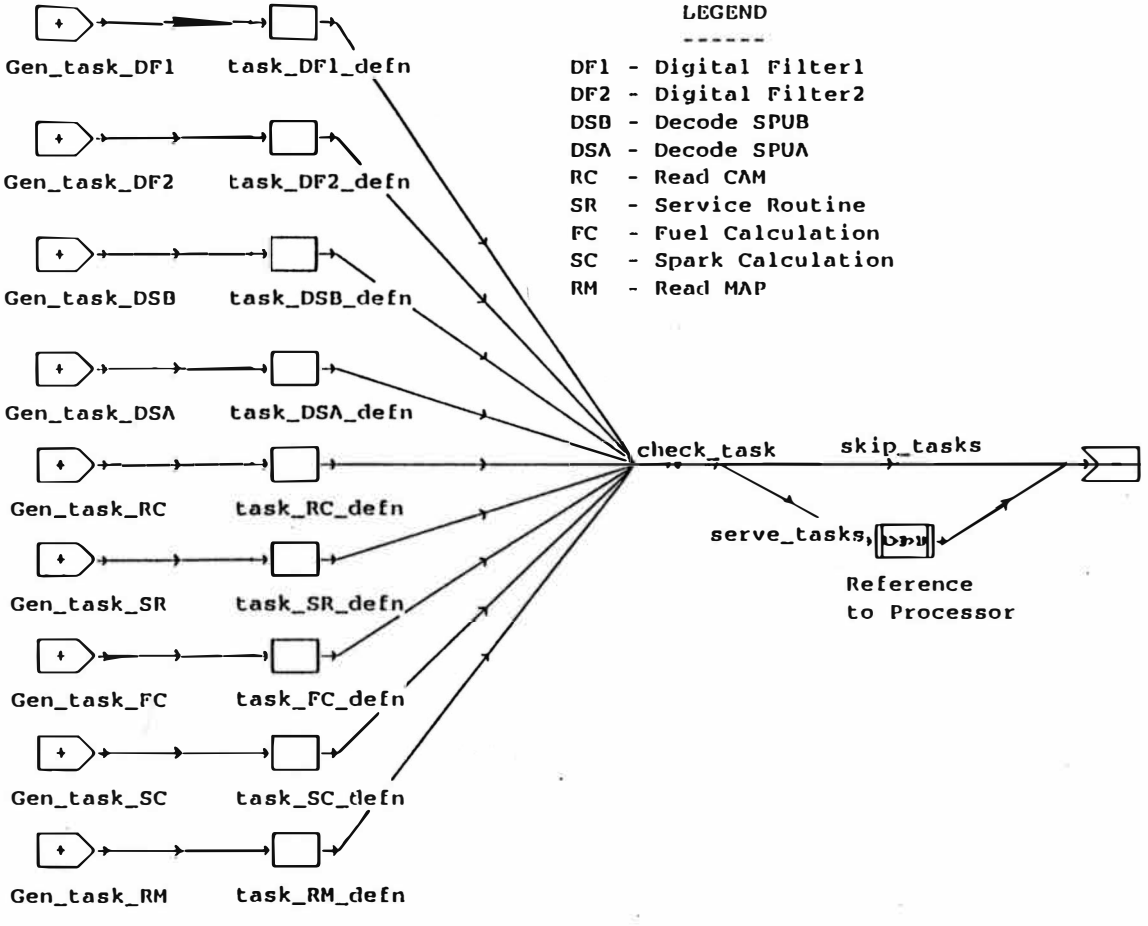

Figure 3. SES Model for Engine Control System.

the Gen_task_* node picks up the required data (e.g., implementation choice, period, deadline etc.) from the task_*_defn node and the node check_task diverts transactions from DF1, DF2 and SC alone towards the processor and all other transactions toward the sink. In this manner, one can simulate all 512 possible system configurations for the engine control module. We include all possible configurations so as to construct a reasonable set of data points in the evaluation of various flexibility metrics. Using the definition given in Chapter II for a_i, p_i, c_i and d_i, we can calculate the simulation time for each system configuration using the formula $a_{max}+2P$ where P is the least common multiple of the periods and a_{max} is the maximum among the activations of all tasks. It should be noted that the same SES model can be used for simulation studies of RGT systems also.

The main module page treats the processor as a reference to another sub-model which is depicted in Figure 4. Every transaction that enters the processor submodel is held at the node activation_delay before reaching the cpu node where the transaction is processed depending on its priority. All transactions exiting the cpu node is checked for deadline violation before reaching the sink. The feas_calculator node collects the required data from the transactions to calculate the value of the metric under consideration for further analysis.

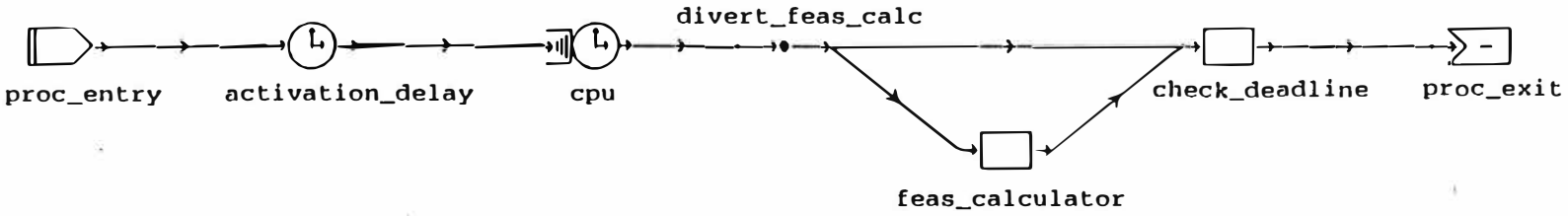

Figure 4. SES Sub-model for the Processor Referred in Engine Control System.

As already pointed out, the analysis we are interested in are primarily used during the initial system design evaluation stage and hence details on overhead due to context switching, task scheduling and pre-emption may not be available. We thus neglect these overhead in our model which also greatly simplifies the realization of the processor in SES simulation. However, these overhead may become a significant factor when the system is actually implemented and therefore may affect the estimated timing requirements. We provide an analysis of potential metrics in handling these factors in the next chapter.

CHAPTER IV

STUDY OF POTENTIAL FLEXIBILITY METRICS

If we knew the precise amount of processing power (such as in terms of average clocks per instruction or MIPS) required to feasibly schedule all tasks and the actual processing power of the given processor, we would be able to precisely predict the system feasibility and its flexibility with respect to the timing requirements. We define ρ as the ratio of the required processing power to the processing power of processor P. Clearly, ρ can be used as a flexibility metric. However, as we have pointed out previously, finding the required processing power is a computationally inhibiting job. Hence, we need to investigate other possibilities of estimating the flexibility of a system.

This chapter focuses on evaluating the effectiveness of several possible metrics for measuring the flexibility. We also propose two new metrics towards predicting the timing behavior and illustrate its improvement over the existing ones. As stated in Chapter II, we use RM algorithm for task scheduling and assume pre-emptive scheme. We also assume that tasks are independent and allocated on a single processor.

Pessimistic Analysis Based Approach

Average Processor Utilization Based Approach

One of the frequently used methods to determine feasibility is via an upper bound on the processor utilization. We would like to investigate if this is an appropriate metric for measuring flexibility. The average processor utilization for a set of n tasks is defined as

$$U = \sum_{i=1}^{n} \frac{c_i}{p_i} \tag{3}$$

Liu and Layland [17] proved that for $d_i = p_i$ and under the worst case phasing, feasibility is guaranteed if the system satisfies (2). We repeat this equation here as (4) for convenience.

$$U = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq n\left(2^{\frac{1}{n}} - 1\right) \tag{4}$$

In some practical systems such as the one described in Chapter 3, where $d_i \neq p_i$, or $a_i \neq 0$ for some τ_i's, (4) is no longer a valid upper bound. A simple modification gives the following worst case feasibility prediction formula.

$$U = \sum_{i=1}^{n} \frac{c_i}{d_i - a_i} \leq n\left(2^{\frac{1}{n}} - 1\right) \tag{5}$$

The inequality given in (5) suggests that an upper bound on ρ can be obtained as

$$\rho^{u_1} = \left[n\left(2^{\frac{1}{n}} - 1\right)\right]^{-1} \sum_{i=1}^{n} \frac{c_i}{d_i - a_i} \qquad (6)$$

Notice that (6) can be treated as a generalized form of (4) since when d_i=p_i and a_i=0, it reduces to (4). Accordingly, we call scheduling based on d_i-a_i as generalized RM (GRM) scheduling. Now, ρ^{u_1} may be considered as a flexibility metric provided that a smaller value of ρ^{u_1} indicates a higher possibility of a system being feasible. Clearly, a system is feasible if $\rho^{u_1} \leq 1$. Careful study is needed when $\rho^{u_1} > 1$ since the systems may or may not be feasible.

We examine the timing behavior for systems with $\rho^{u_1} > 1$ as follows. Given the example task systems and a large number of RGT systems, we compute ρ^{u_1} for each system and then simulate (based on GRM scheduling) to determine its feasibility. Figure 5 depicts the results of such an approach. In this graph, the results obtained for various processor MIPS in the engine control system example have been combined. The percentage feasible systems over a given range indicates the ratio of the actual number of systems feasible to the total number of systems whose ρ^{u_1} values fell in that range. From Figure 5, it is not difficult to note that even for an upper bound value of around 2, more than 80% of the systems are actually feasible. This shows the over-estimation of the schedulability of the tasks

when we use the worst-case prediction approach for determining the feasibility. Another interesting yet somewhat counter-intuitive observation is that there are ample number of cases where systems with higher values of ρ^{u_1} are more likely to be feasible than systems with lower values of ρ^{u_1}. This indicates that ρ^{u_1} as defined in (6) is not an appropriate measure for flexibility. That is, by simply computing ρ^{u_1}'s of two systems, one cannot

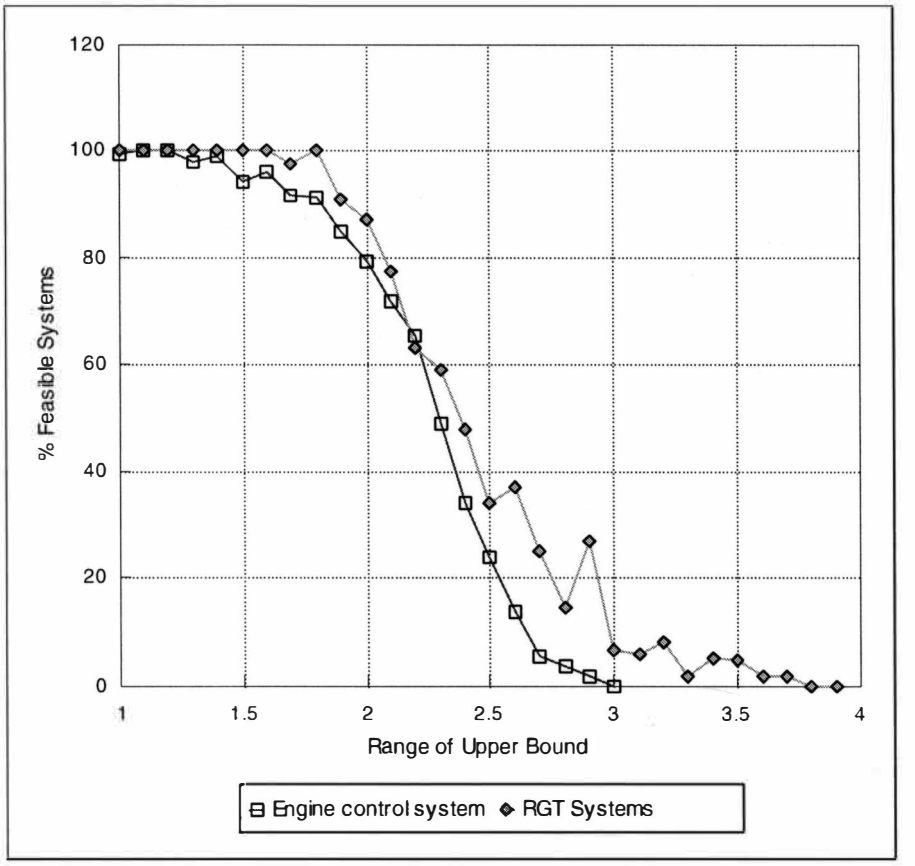

Figure 5. Graph Showing the Likelihood of Systems Being Feasible When Upper Bound as Defined in (6) is Used as a Predictor.

decide if one system is more flexible (or more likely to be feasible) than the other.

Response Time Based Approach

Another worst-case analysis based technique is the fixed-point iteration method discussed in [14] and developed further by Yen and Wolf in [23]. It finds the worst case response time of τ_i as follows. Initially let

$$X = \left\lceil c_i / (1 - \sum_{j=1}^{i-1} \frac{c_j}{p_j}) \right\rceil \quad and$$

$$g(X) = c_i + \sum_{j=1}^{i-1} c_j \lceil X / p_j \rceil$$

Then,

while $(X < g(X))$
{
$X=g(X)$;
$g(X) = c_i + \sum_{j=1}^{i-1} c_j \lceil X / p_j \rceil$;
}

It follows that $g(X)$ converges to the worst case response time of τ_i, r_i. Note that the tasks are assumed to be arranged in the decreasing order of priorities. We can calculate the upper bound for a system based on the worst case response time as

$$\rho^{u_2} = \frac{r_i + a_i}{d_i} \qquad (7)$$

If $\rho^{u_2} \leq 1$, then the systems are guaranteed to be feasible. If $\rho^{u_2} > 1$, then an analysis as for ρ^{u_1} is needed. The results are shown in Figure 6. Clearly, the fixed-point iteration technique can give better prediction results than (6). However, if the number of tasks in a system is large, this method can be quite expensive. Furthermore, the fixed-point iteration approach assumes the worst-case phasing which again, may lead to over specification.

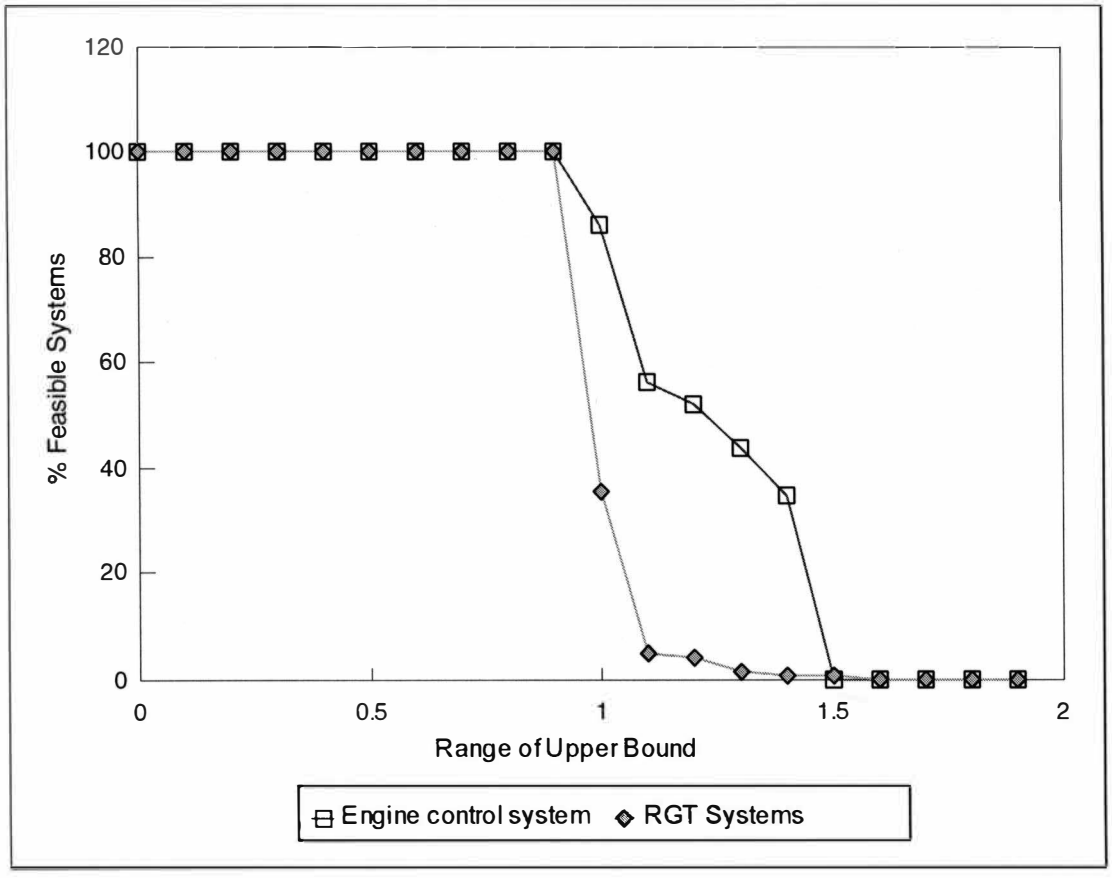

Figure 6. Graph Showing the Likelihood of Systems Being Feasible When Upper Bound as Defined in (7) is Used as a Predictor.

Yen and Wolf proposed an elegant method for handling the case where not all tasks are activated at the same time. They model this by dummy tasks with data dependencies [23]. Nonetheless, their algorithms can be computationally expensive considering that many system configurations may need to be investigated.

A Lower Bound Based Approach

The inferences on the results based on the upper bound based techniques suggest that we may investigate its counter-part, the lower bound based approach. It is well known that a task set is definitely infeasible if the average processor utilization defined in (1) is greater than 1. This can be used to impose a lower bound on ρ. That is,

$$\rho^{l_1} = \sum_{i=1}^{n} \frac{c_i}{p_i} \qquad (8)$$

Consequently, if $\rho^{l_1} > 1$, the system is infeasible. We would like to see whether the value of ρ^{l_1} in the range [0,1] can indicate the flexibility of a system. We can consider ρ^{l_1} to be good flexibility metric if systems with a lower value of ρ^{l_1} indicate a higher percentage of feasible systems. In other words, a similar analysis as for upper bound when carried out for ρ^{l_1} should result in a monotonically decreasing graph. The results de-

picted in Figure 7 indicates that this is not true since a peak can be noticed for the RGT systems. However, it is relatively a reliable measure than the upper bound since the results for ρ^{l_1} show a better monotonic trend than that of ρ^{u_1}. Hence one may consider ρ^{l_1} defined in (8) as a potential candidate for predicting flexibility.

A significant drawback of using ρ^{l_1} as a metric is that the percentage feasible systems drops quickly as the value of ρ^{l_1} increases. This means that when we have a system with ρ^{l_1} relatively low (say 0.5), it is

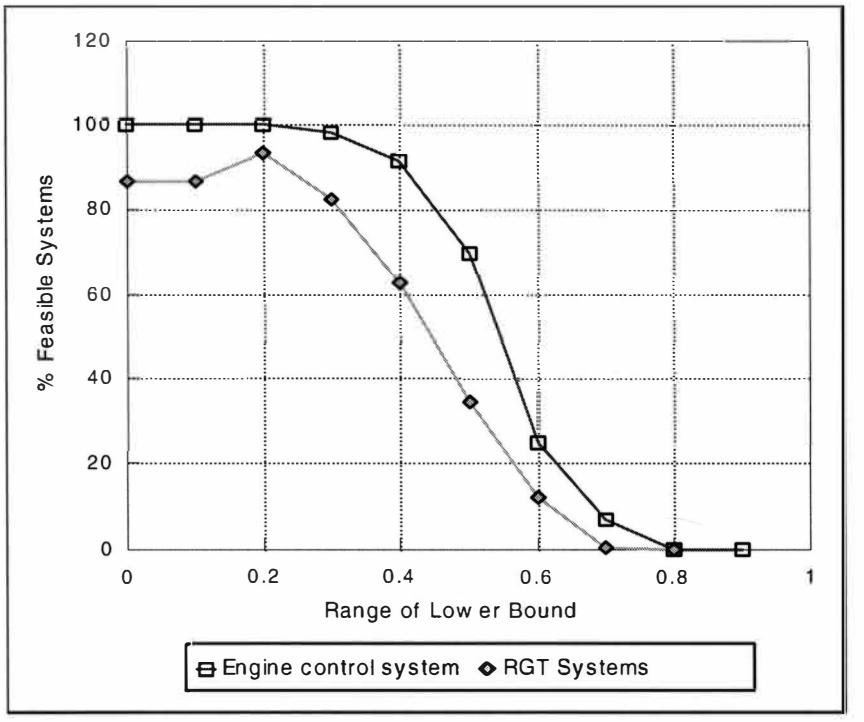

Figure 7. Graph Showing the Likelihood of Systems Being Feasible When Lower Bound as Defined in (8) is Used as a Predictor.

not clear whether it is worthwhile to investigate such system configuration further. In addition, using ρ^{l_1} as the only timing analysis parameter may fail to detect certain obviously infeasible systems. For example, consider a three system shown in Table 4. The value of ρ^{l_1} for this system is 46.7% which suggests that this system could be feasible. However, it is easy to find from the timing parameters that the third task would never meet its deadline since the minimum time it has to wait for higher priority tasks is 7 and hence the system is infeasible. Thus, ρ^{l_1} cannot be used as the only measure to indicate the flexibility of a system.

Table 4

A Sample System to Illustrate the Deficiency of Loose Lower Bound

i	a_i	p_i	c_i	d_i
1	0	10	4	6
2	0	30	3	10
3	0	120	8	14

A Tighter Lower Bound Based Approach

We have seen that ρ^{l_1} in (8) is close to being an appropriate candidate. In [8], a tighter lower bound on ρ was given , but no sufficient data were provided to illustrate the effectiveness of this bound. We will give a

slightly modified lower bound calculation and study its behavior with respect to timing prediction. Consider a task set of n tasks. For the first execution request of τ_i, any task requests that have deadlines preceding d_i must be completed before d_i. Thus the total computation requirement between [0, d_i] includes at least satisfying all these task requests. We state the following lemmas that can be used to calculate the number of task requests in intervals $[a_j,d_i]$ for $d_j \le d_i$ and $[a_i,d_i]$.

Lemma 1: *For i tasks that are arranged in the ascending order of their deadlines, define*

$$k_j \equiv \begin{cases} \lceil (d_i - a_j)/p_j \rceil & if \lfloor (d_i - a_j)/p_j \rfloor . p_j + d_j \le d_i \\ \lfloor (d_i - a_j)/p_j \rfloor & otherwise \end{cases}$$

then the i tasks cannot be feasibly scheduled if

$$\sum_{j=1}^{i} \frac{k_j . c_j}{d_i - a_{\min}} > 1 \quad where \quad a_{\min} = \min_{1 \le j \le i} a_j$$

Proof: Let τ_j (j=1,2,...i-1) represent all tasks with priority higher than τ_i. It is clear that for the first execution request of task τ_i, any task requests from τ_j with deadlines prior or equal to d_i, must be completed before d_i in order that the system is feasible. Therefore, the total time the processor is occupied by all higher priority tasks than task τ_i may be given as $k_j.c_j$, where k_j is the minimum number of times task τ_j needs to be executed in

the time interval $[a_j, d_i]$.

The value of k_j can be calculated by considering those requests of τ_j which start before d_i. The number of requests for τ_j within the interval $[a_j, d_i]$ is $\lceil (d_i - a_j)/p_j \rceil$. We need to decide whether the last request for task τ_j as determined by $\lceil (d_i - a_j)/p_j \rceil$ must be finished before d_i. Notice if $\left(\lfloor (d_i - a_j)/p_j \rfloor\right) p_j + d_j \le d_i$ then, the last request of task τ_j has to be completed before d_i. Otherwise, this request need not be finished before d_i and hence only $\lfloor (d_i - a_j)/p_j \rfloor$ number of τ_j requests need to be considered.

It follows that the amount of execution time required to process τ_i's initial request as well as all the higher priority tasks' requests within $[a_{min}, d_i]$ must be at least $\sum_{j=1}^{i} k_j.c_j$. Then if, $\sum_{j=1}^{i} k_j.c_j > d_i - a_{\min}$, the processor would not be able to feasibly schedule the tasks within $[a_{min}, d_i]$. Hence the system is infeasible. §

Lemma 2: *For i tasks that are arranged in the ascending order of their deadlines, let k_j be the same as that defined in Lemma 1. Define*

$$h_j = \begin{cases} k_j - \left\lceil \dfrac{a_i - a_j}{p_j} \right\rceil & \text{if} \quad a_j < a_i \\ k_j & \text{otherwise} \end{cases}$$

Then, the i tasks cannot be feasibly scheduled if

$$\sum_{j=1}^{i} \frac{h_j . c_j}{d_i - a_i} > 1$$

Proof: The proof for this can be obtained along the same lines as Lemma 1. It is clear that $h_j = k_j$ if $a_j \geq a_i$ since the release time for task τ_j is a_j and within the interval $[a_i, a_j]$, there will be no requests from τ_j. Now, if $a_j < a_i$, the number of requests of τ_j in the interval $[a_i, d_i]$ may be obtained as the number of requests in the interval $[a_j, d_i]$ minus the number of requests in the interval $[a_j, a_i]$. §

Based on these lemmas, we can now define a tighter lower bound as given below.

$$\rho^{l_2} = \max_{i=1}^{n} \left\{ \sum_{j=1}^{i} \frac{k_j . c_j}{d_i - a_j}, \quad \sum_{j=1}^{i} \frac{h_j . c_j}{d_i - a_i} \right\} \tag{9}$$

To investigate the effect of ρ^{l_2}, as given in (9), similar analysis as those for upper bound and loose lower bound was performed. Figure 8 depicts the results. Clearly, the graph is monotonic and hence is a good measure of the flexibility of the system. It is interesting to note that the percentage feasible systems is quite high even when ρ^{l_2} is relatively large (e.g. 0.5). Thus, ρ^{l_2} is a quite reliable predictor for feasibility. Furthermore, let us define critical excess requirement ratio as $\rho^c = 1 - \rho^{l_2}$.

The value of ρ^c provides an estimate of the additional load the proc-

essor could handle after meeting the current task specifications, which is a natural measure of flexibility given that ρ^{l_2} is monotonic. Hence, during system architectural exploration, we can use ρ^{l_2} or ρ^c as a flexibility metric and study the trade-off of flexibility against cost, power consumption etc.

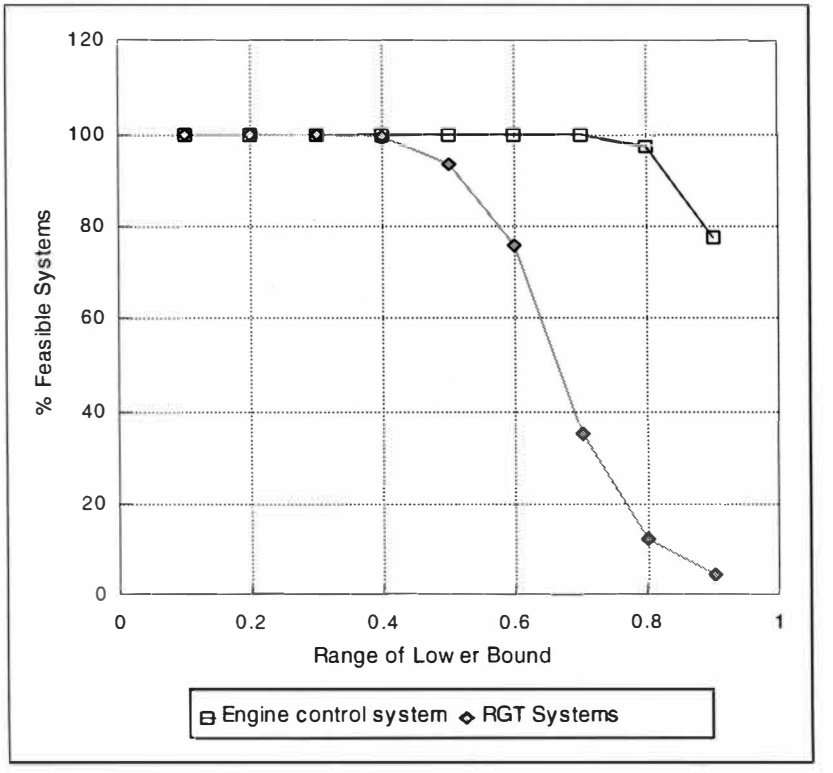

Figure 8. Graph Showing the Likelihood of Systems Being Feasible When Lower Bound as Defined in (9) is Used as a Predictor.

Though using ρ^{l_2} can be an effective means to study potential design candidates in the architectural design exploration of real time sys-

tems, this analytical technique may become computationally expensive when the number of tasks grow larger. It would be desirable to have a metric that is less computationally involved.

A Feasibility-Factor Based Approach

In [8], the authors introduced a feasibility measure called feasibility factor based on the upper and lower bounds of throughput requirements. We would like to generalize the definition of feasibility factor and study its behavior. We define the feasibility factor as

$$\lambda \equiv \frac{1-\rho^{l}}{\rho^{u}-\rho^{l}} \quad if\ \rho^{u} \neq \rho^{l} \qquad (10)$$

where ρ^{l} could be any of the two lower bounds discussed earlier. Notice if $\rho^{u} = \rho^{l}$, we have a precise prediction of ρ. As illustrated in Figure 9, a set of tasks allocated on a processor are feasible if $\lambda \geq 1$ and they are not feasible if $\lambda < 0$. For $0 \leq \lambda < 1$, the feasibility of the system cannot be predicted solely based on λ and needs to be carefully analyzed.

To examine the behavior of λ, we performed a analysis similar to those done for previous metrics. Since λ can be computed based on any given formula for ρ^{l} and ρ^{u}, we have to obtain different λ values by using ρ^{l_1} and ρ^{l_2}. The data obtained are summarized in Figure 10. Notice that

this metric is monotonic for both ρ^{l_1} and ρ^{l_2} based calculations. Using our previous reasoning, λ is a reliable predictor for feasibility. Furthermore, it can be considered as an estimate of critical excess requirement ratio. Instead of using $1-\rho^l$ directly, the value of $1-\rho^l$ is scaled by $(\rho^u - \rho^l)$. Such a

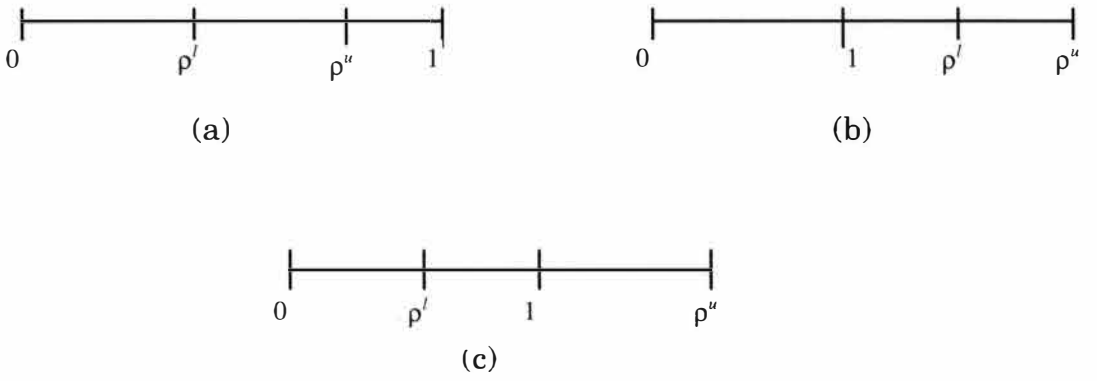

Figure 9. Ranges of Upper and Lower Bounds Showing Schedulability Based on λ. (a) Definitely Schedulable System ($\lambda \geq 1$), (b) Definitely Not Schedulable System ($\lambda < 0$), (c) Further Analysis is Needed to Determine Schedulability ($0 \leq \lambda < 1$).

scaling gives better estimation since it includes the effect of ρ^u. Therefore, we can use λ directly to measure the flexibility of a system architecture. A larger value of λ indicates that the system is relatively more feasible. Note that when ρ^{l_2} is used, the likely-hood of the system being feasible is higher, compared with the case based on ρ^{l_1}. Of course, the computational needs for the one based on ρ^{l_2} is larger. Nevertheless, it is up to the dis-

cretion of the system designer to choose either of these methods, since both of them exhibit suitable characteristics.

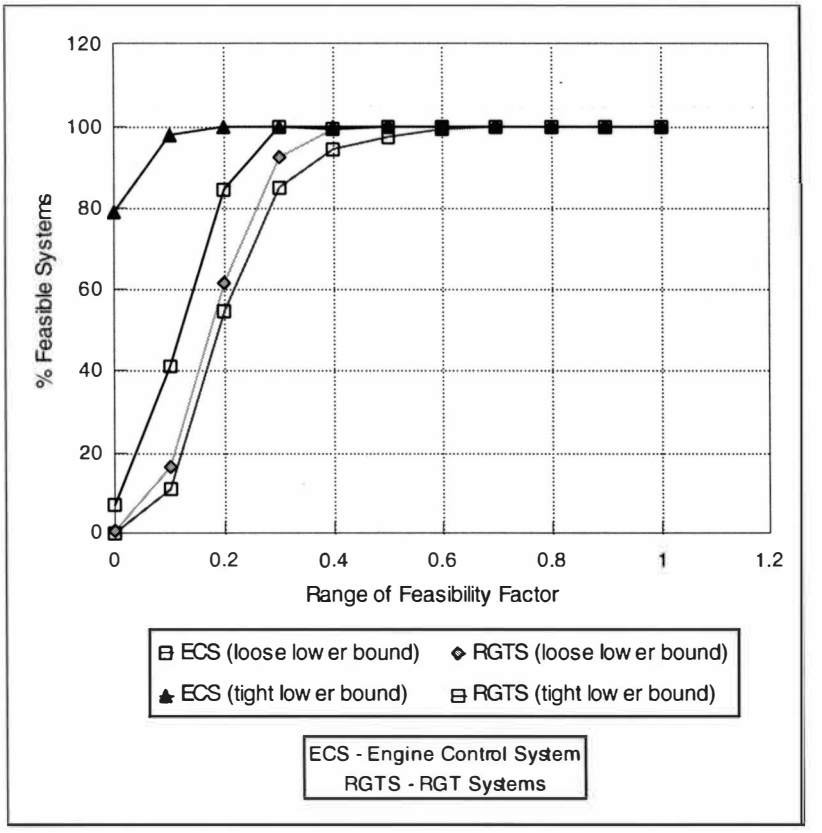

Figure 10. Graph Showing the Likelihood of Systems Being Feasible Vs. λ.

Behavior of Potential Metrics on Processor Overload

We have seen that the lower bounds and feasibility factor to are potential candidates for predicting the flexibility of a system. The analysis we have done thus far concerns systems which are likely to be feasible. In

this section, we will discuss our results for analysis done on systems which are guaranteed to be feasible.

As we have already pointed out, in real-time embedded systems, the system specifications may not be fully available at the architectural design stage. Quite often, the timing parameters such as c_i and d_i are based on some kind of estimates and therefore, they are likely to change when the system is actually implemented. Thus, we would like to see the effectiveness of the metrics considered previously in reflecting the change in timing requirements due to the change in the timing parameters of the system.

From the RGT systems that we generated, we collect all feasible task sets as determined by the simulation and perturb the value of c_i for each task in a task set in order to achieve a specified overload (calculated based on the metric under consideration) on the processor. We would like to find out the possibility of a system being feasible under the overload situation. We again perform the same analysis as we did in the previous section. The results are depicted in Figures 11, 12 and 13. Clearly, the behavior of the upper bound is highly unpredictable. The lower bound based on processor utilization performs better than the upper bound in reflecting the processor's capability of handling the overload. However, as mentioned earlier, we measure the goodness of a metric in terms of the mono-

tonicity of the curve. Clearly, the results obtained for the lower bound are not monotonic. We therefore conclude that the feasibility factor is a better candidate in reflecting the change in timing requirements of an RTES as evidenced from Figure 13.

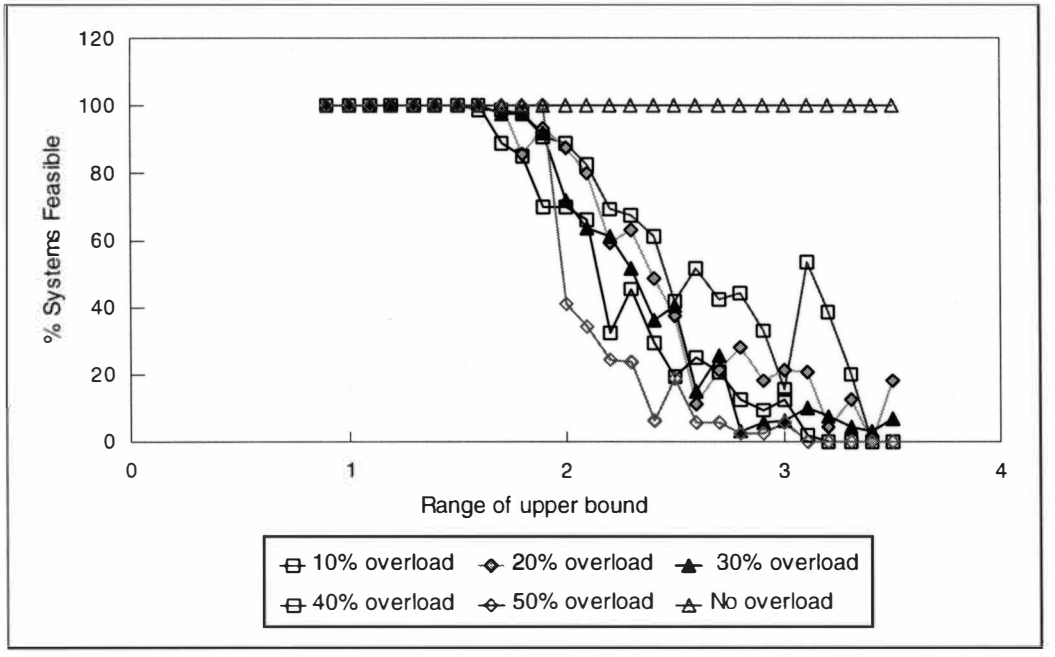

Figure 11. Behavior of Upper Bound (ρ^{u_1}) for Various Levels of Processor Overload.

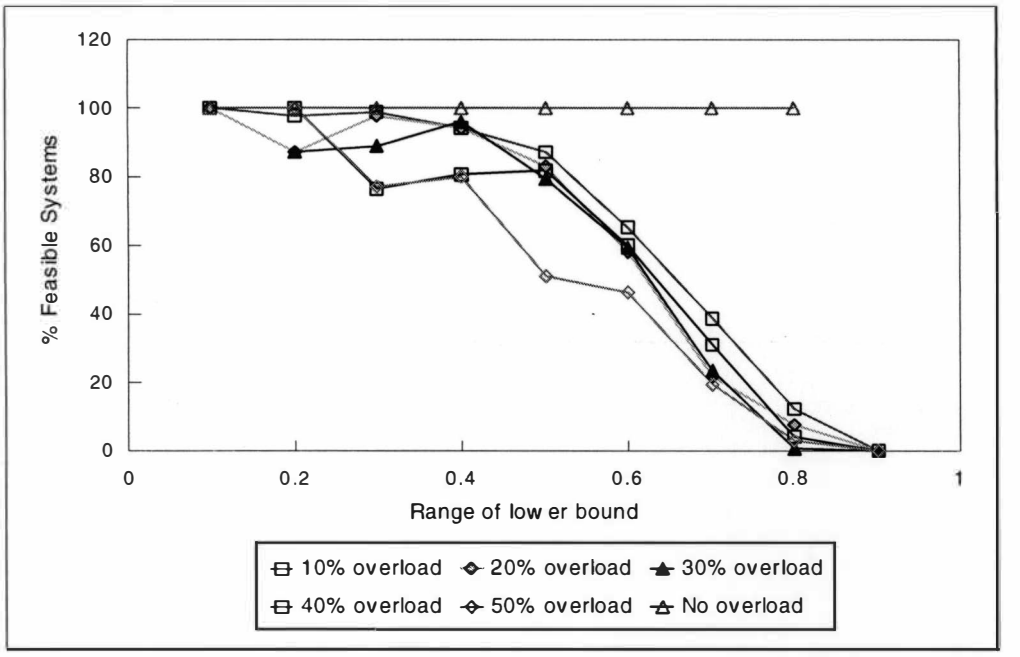

Figure 12. Behavior of Lower Bound (ρ^{l_1}) for Various Levels of Processor Overload.

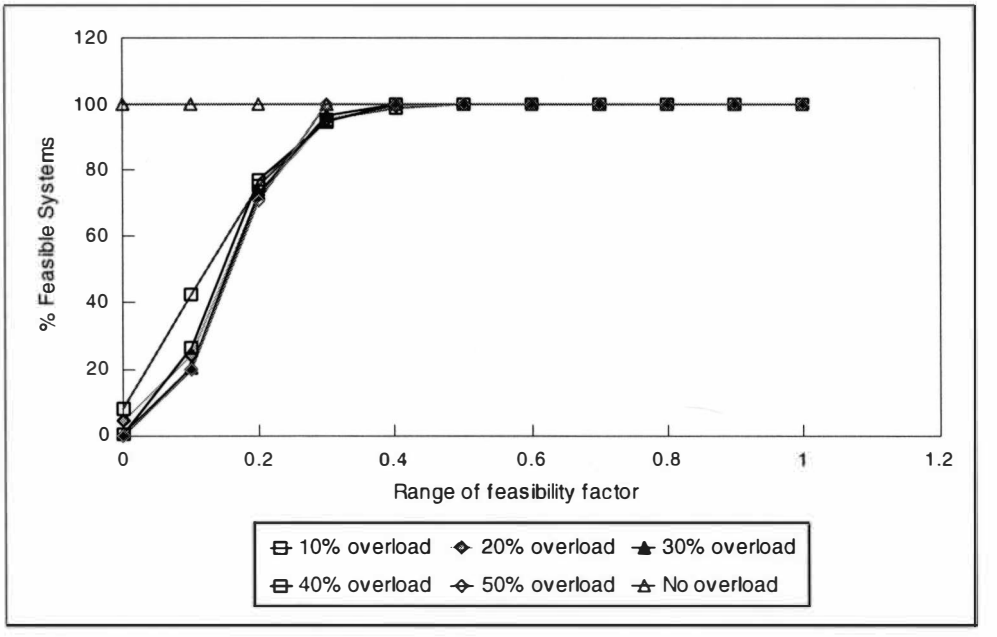

Figure 13. Behavior of Feasibility Factor for Various Levels of Processor Overload.

CHAPTER V

CONCLUSION AND FURTHER WORK

In this thesis, we motivated the need for a flexibility metric for an efficient analysis of potential design candidates in the architectural design exploration of real-time embedded systems. We have shown that some intuitive measures such as the upper bound based on average processor utilization are highly pessimistic and often leads to over-specification of timing requirements. We have also seen that worst case response time based techniques may also lead to over-specification. Further the response time based techniques cannot be easily extended to systems where tasks have deadlines exceeding their periods and the fixed point iteration approach may become computationally expensive. It has also been exemplified that using the lower bound approach based on processor utilization as the only measure for obtaining the timing requirements may fail to detect certain obviously infeasible systems.

We have identified that the tighter lower bound and feasibility factor based approaches are effective in quantifying the timing requirements of a RTES. The feasibility factor based approach is also shown to perform better in reflecting the changes in timing requirements of a sys-

tem. These metrics can be reliably used to compare potential design candidates at the architectural system design stage. They can also be used as both a constraint as well as an attribute in hardware/software partitioning of real time systems [8]. In the constraint case, we eliminate the infeasible solutions and in the attribute case, we can efficiently evaluate systems modeled at the configuration level.

The limitation of this work is that we have considered only uniprocessor systems and assumed that the software tasks running on the processor do not have any dependency. We have also assumed that the deadlines of the tasks do not exceed their periods. We intend to expand our work to include these cases. Task dependency can be modeled as activation times between tasks.

Another approach that we are currently investigating to predict the schedulability of a system is the integer linear programming (ILP) approach. Specifically, we use the timing parameters such as c_i, d_i, p_i, and a_i to develop a objective function which has to be optimized under certain constraints. The "*level-i*" busy period concept proposed in [13] may be effectively used to model the scheduling problem as an ILP problem.

BIBLIOGRAPHY

[1] P. Athanas and H.F. Silverman, "Processor reconfiguration through instruction-set metamorphosis," *Computer*, vol. 26, no. 3, 1993, 11-18.

[2] N. Audsley, A. Burns, M. Richardson, K. Tindell and A.J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, 1993, 284-292.

[3] J. Beck and D. Siewiorek, "Automated processor specification and task allocation for embedded multicomputer systems: The packing-based approaches", *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing,* 1995, 44-51.

[4] K. Buchenrieder and C. Veith, "CODES: a practical concurrent design environment", *Handouts from International Workshop on Hardware-Software Codesign*, 1992.

[5] S. K. Dhall, and C. L. Liu, "On a real-time scheduling problem", *Operations Research,* Vol. 26, No. 1, 1978, 127-140.

[6] M. R. Garey, and D. S. Johnson, "Complexity results for multiproces sor scheduling under resource constraints", *Society for Industrial and Applied Mathematics Journal of Computing,* 1975.

[7] M.R. Garey, D.S. Johnson, B.B. Simon and R.E. Tarjan, "Scheduling unit-time tasks with arbitrary release times and deadlines", *SIAM Journal on Computing,* vol. 10, no. 2, 1981, 256-269.

[8] X. Hu and J.G. D'Ambrosio, "Configuration-level hardware/software partitioning for real-time embedded systems", to appear in *Journal of Design Automation for Embedded Systems.*

[9] X. Hu, J.G. D'Ambrosio, B.T. Murray, and D. Tang, "Codesign of Architectures for Automotive Powertrain Modules", *IEEE Micro,* 1994, 17-25.

[10] M. Joseph and P. Pandya, " Finding response times in a real-time system", *Computer Journal*, vol. 29, no.5, 390-395.

[11] S. Kumar, J.H. Aylor, B.W. Johnson and W.A. Wulf, "Object-oriented techniques in hardware design", *Computer,* vol. 27, no. 6, 1994, 64-70.

[12] E. L. Lawler, and C. U. Martel, "Scheduling periodically occurring tasks on multiple processors", *Information Processing Letters*, Vol. 12, No. 1, 1981, 9-12.

[13] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines", *Proceedings of the 11th Real time systems symposium*, 1990, 201-209.

[14] J. P. Lehoczky and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems", *Proceedings of Real-Time Systems Symposium,* 1992, 110-123.

[15] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling al gorithm: exact characterization and average case behavior", *Proceedings of the 1989 IEEE Real-time System Symposium*, 1989, 166-171.

[16] J. Y-T Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks", *Performance Evaluation,* vol.2, 1982, 237-250.

[17] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", *Journal of the Association for Computing Machinery,* vol. 20, no. 1, 1973, 46-61.

[18] A.K. Mok, "Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment", PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1993.

[19] R. R. Muntz, and E. G. Coffman, Jr., "Preemptive scheduling of real-time tasks on multiprocessor systems", *Journal of the ACM*, Vol. 17, No. 2, 1970, 324-338.

[20] R. Rajkumar, L. Sha, and J. P. Lehoczky, "Real-time synchronization protocols for multiprocessors", *Proceedings of 1988 IEEE Real-time Systems Symposium*, 1988, 259-269.

[21] K. Ramamritham, and J. A. Stankovic, "Dynamic task scheduling in distributed real-time systems", *IEEE Software*, Vol. 1, No. 3, 1984, 65-75.

[22] M.B. Srivastava and R.W. Brodersen, "Rapid-prototyping of hardware and software in a unified Framework", *Proceedings of International Conference on Computer-Aided Design,* 1991, 152-155.

[23] T.-Y. Yen and W. Wolf, "Performance estimation for real-time distributed embedded systems", *Proceedings of the International Conference on Computer Design (ICCD'95),* 1995, 64-69.

[24] T.-Y. Yen and W. Wolf, "Communication synthesis for distributed embedded systems", *Proceedings of the International Conference on Computer-Aided Design*, 1995, 288-294.

[25] J. Zalewski, "Real-Time Systems Glossary", *Dept. of Computer Science, The University of Texas of the Permian Basin*, 1993.

[26] W. Zhao, K. Ramamritham and J. Stankovic, "Preemptive scheduling under time and resource constraints", *IEEE Transactions on Computers,* vol. 36, no. 8, 1987, 949-960.