CAISARTS: A Tool for Real-Time Scheduling Assistance

M. Humphrey and J.A. Stankovic

CMPSCI Technical Report 96-07

January 1996

CAISARTS: A Tool for Real-Time Scheduling Assistance*

Marty Humphrey and John A. Stankovic
Department of Computer Science
University of Massachusetts, Amherst, MA 01003
Email: HUMPHREY@CS.UMASS.EDU

Abstract

CAISARTS (Conceptual, Analytical, and Implementation Scheduling Advice for Real-Time Systems) is a rule-based system used by real-time application designers to obtain expert assistance for all aspects of the design related to scheduling: granularity of tasks, allocation of tasks, choice and analysis of scheduling paradigm, analysis of overheads of particular operating systems and scheduling paradigms, and code templates for tasks. The rule base is partitioned; subsets of the rule base can be selected for firing, thus enabling the user to ask CAISARTS for advice and analysis relevant for different phases of the design. In contrast to existing real-time tools, CAISARTS attempts to cover the entire design process related to scheduling without focusing on, for example, solely schedulability analysis. A unique feature of CAISARTS is that its rule base is extensible by the user—a graphical interface is used to add new rules as new real-time results are identified. Challenges in the design of the initial rule set include how to design and partition the rule base so that it can be both easily modifiable and readily usable by the user in choosing rules to fire; how to encode rules that are inherently contradictory; how to encode ambiguous knowledge; and how to make the rules both comprehensive and precise. The effectiveness of CAISARTS is shown through its use on a representative distributed real-time system scenario—CAISARTS generates analysis and advice that is consistent with existing analysis of this environment. Further capabilities are shown when CAISARTS is applied to this environment after simplifying assumptions have been removed and end-to-end constraints are added.

1 Introduction

Real-time systems are very complex. Some of the most critical aspects of such systems are scheduling and resource allocation. In these areas it is easy to make errors; when they occur it is sometimes very difficult to detect the causes. Sometimes the errors are logical errors made at the algorithmic level, other times the errors are due to erroneous methods of including or not

*This work is funded in a joint Industry-University project by Advanced System Technologies, Englewood, CO, under SBIR Contract No. N60921-93-C-0178.

including system implementation overheads into the analysis, and yet other times the errors are due to implementation (coding) errors. For example, a designer may use an algorithm to compute an optimal static real-time schedule for a three-processor multiprocessor. Later, the designer may decide to add a fourth processor and find that the system now misses deadlines. This is due to Richard's anomalies [Graham, 1969] – an algorithmic level misunderstanding. In Rate Monotonic Analysis [Klein *et al.*, 1993] it is often quite easy to compute schedulability bounds considering task execution times and periods. However, accurate incorporation of system level overhead for items such as putting a task to sleep and re-invoking it at the next period, overheads dealing with waits for resources, etc. may not be properly included in the analysis. The designer may then think that all deadlines will be met but finds that unconsidered overhead causes deadlines to be missed. Finally, simple coding errors may crop into the implementation of even standard scheduling algorithms and associated OS code.

CAISARTS (Conceptual, Analytical, and Implementation Scheduling Advice for Real-Time Systems) is a tool that aids a real-time system designer in all phases of the design process related to scheduling. CAISARTS is based on the observation that each of the three phases of managing tasks and resources in a real-time system—allocation, scheduling, and dispatching—cannot be dealt with in total isolation since the mechanism used in one phase may greatly affect the performance of others. CAISARTS contains two parts: an environment modeling system and an inference engine. The user models the environment by using primitive objects such as tasks, hardware and shared resources. The rule base of CAISARTS contains real-time system scheduling knowledge in the form of IF-THEN rules. The rule base is partitioned hierarchically into *rule containers*, so that a user may select and use different portions of the rule base at different points in the design process, thus enabling the user to ask for only the most pertinent advice. The rule base incorporates rules, rationale, analysis and code templates.

A unique feature of CAISARTS is that the user may add a new rule to the rule container or create a new rule container as new knowledge is identified—real-time system knowledge is not solely "compiled" into the tool by the developers of CAISARTS. CAISARTS is also capable of representing the often-conflicting approaches of the real-time community in the form of conflicting rules. While contradictory rules are generally a problem for a rule-based system, the approach in CAISARTS is to fully encode the state of the art in real-time research, even if this results in contradictory rules. If contradictory rules fire, the user is presented with conflicting approaches, *with the explicit warning to the user that these are conflicting approaches*, and the user has the right to choose the better approach for his/her needs. Another strength of CAISARTS is its ability to represent real-time system scheduling knowledge in varying forms of precision: a rule can be written such that it fires only on precise input conditions (which means that the advice is very specific to the input environment), or can be written to fire on very general conditions and provide very general, high-level advice. Rules that provide general advice in a particular area can be used to reflect that the state of the art has not been developed, as is the case with many real-time system scenarios. The ability to write such rules is important, because a specific goal of CAISARTS is to capture and describe what precisely is known, and what precisely is not known, related to scheduling in real-time systems.

The goal of this paper is to show that the approach taken in CAISARTS is well-suited for encoding real-time system scheduling advice. The discussion of scheduling knowledge encoded in CAISARTS focuses on general distributed systems and uniprocessor static priority scheduling, because these rules have been developed to a greater degree than rules in other areas,

such as code templates. The intent of this paper is *not* the presentation of a comprehensive set of rules that encode knowledge related to scheduling in real-time systems, but rather a discussion of difficulties and concerns that arose as an initial set of rules were encoded, and a discussion of CAISARTS as a general framework for providing real-time system advice related to scheduling. To illustrate the utility of CAISARTS, we describe its application on a standard real-time benchmark published in the literature [Molini *et al.*, 1990] (see Section 5). Using this benchmark we demonstrate that CAISARTS provides advice and analysis for many stages of a design process; the analysis provided by CAISARTS is consistent with published analysis [Molini *et al.*, 1990]. In a second part of the validation, the benchmark is extended to produce a workload that includes distributed end-to-end communication and timing constraints. The use of CAISARTS in this environment demonstrates its value for distributed real-time systems.

This paper is organized as follows. Section 2 covers tools described in the literature that are related to the approach and design of CAISARTS. Section 3 contains an overview of CAISARTS, which includes a description of how the environment is modeled and how the rules are invoked. Section 4 covers topics related to the design and implementation of the rule base. Section 5 illustrates the use of the tool on the real-time benchmark. The conclusions of the paper are in Section 6.

2 Related Work

Various tools exist to aid the real-time system developer. In general, these tools approach a very specific area of real-time system design. This philosophy contrasts that of CAISARTS, in which both the breadth and depth are important. CAISARTS compliments many of these tools, as it is conceivable that any tool designed for a particular subsystem can be incorporated into CAISARTS—if a tool exists that provides in-depth analysis that is not directly the scope of CAISARTS or has not been provided by CAISARTS itself, then rules can be added to CAISARTS that instruct the user to investigate the use of the other tool.

2.1 Scheduler 1-2-3

Important early work with the design and analysis of real-time systems includes Scheduler 1-2-3 [Tokuda and Kotera, 1988]. As part of a real-time toolset for ARTS and later Real-Time Mach, Scheduler 1-2-3 analyzes the correctness of timing requirements at design time. Scheduler 1-2-3 is a window-based system for describing and analyzing real-time task sets but focuses on fixed priority scheduling under Rate Monotonic priority assignment. Schedulability analysis is also provided by simulation techniques.

CAISARTS is different than Scheduler 1-2-3 in many ways. CAISARTS contains a richer task model, which includes resource requirements, precedence constraints between tasks, end-to-end requirements, and task semantics (hard vs. soft vs. no deadline, value of tasks). The focus in CAISARTS is on providing *advice* at a variety of abstraction levels, whereas Scheduler 1-2-3 is for schedulability analysis. The real-time system knowledge of CAISARTS is extensible by the user via the rule interface. Also, CAISARTS incorporates multiprocessor and distributed systems. It should be noted that the simulation capability of Scheduler 1-2-3 is currently more extensive than CAISARTS. In the future, simulation techniques will be added to CAISARTS.

2.2 PERTS

The originators of PERTS (Prototyping Environment for Real-Time Systems) [Liu *et al.*, 1993] observe that real-time systems are traditionally constructed by first developing the application software, and then validating the timing constraints by using *ad hoc* techniques and extensive simulation. A goal of PERTS is to provide an extensible library of scheduling algorithms, resource access control protocols, and communication protocols in the form of reusable software modules. PERTS includes a rich set of graphics routines that enable the user to quickly determine the results of schedulability analysis. The PERTS model of the environment is richer than the one adopted in Scheduler 1-2-3; the environment model in PERTS includes a complex resource modeling capability.

The goal of PERTS is to provide analysis and reusable software prototypes, but without specifically attempting to provide advice. PERTS is designed to support the evaluation of new and competing designs; it is not meant to directly provide advice for choosing among a set of alternatives. In other words, the manner in which PERTS operates is that advice can be implicitly generated by PERTS if the user iteratively evaluates different configurations and options, after which the user selects the best option.

There are notable differences between CAISARTS and PERTS. Fundamentally, the nature of the tools is different: CAISARTS provides text-based advice on many topics related to the schedulability (such as how to select a particular implementation approach, how to incorporate overheads, and what low-level OS primitives should be avoided), while PERTS provides graphics-based schedulability analysis. CAISARTS can explicitly make the user aware that the environment and requirements as modeled by the user can be approached according to different methodologies. This is important in situations where different parties in the real-time system community advocate different approaches. Options that must be manually selected in PERTS are often automatically deduced by CAISARTS. An example of this is that the user of PERTS must first select a particular fixed priority scheduling policy (either Rate Monotonic or Deadline Monotonic) and a particular synchronization protocol (Priority Ceiling Protocol or Stack-Based Protocol) in order for PERTS to perform schedulability analysis. In CAISARTS, the selection of the priority assignment policy is automated, guided by the matching of the environment as input by the user and well-known optimality conditions of each priority scheme. In other words, the user of CAISARTS does not need to know the specific optimality conditions of various algorithms. This concept generalizes to other type of advice as well.

2.3 iRAT

*i*RAT [Int, 1994] is a window-based engineering tool formulated on the analytical techniques of fixed priority scheduling theory and Rate Monotonic Analysis. *i*RAT consists of a schedulability engine, transformation and analysis dialogs, comprehensive diagnostics, and customized reports. The prime difference is that *i*RAT provides Rate Monotonic Analysis techniques and CAISARTS attempts to provide advice in broader areas beyond schedulability.

2.4 Tools for Modeling Real-Time System Architectural Components

Tools have been developed to model various architectural features used in real-time systems. A representative example of these tools models bus scheduling policies [Kettler *et al.*, 1995b].

This tool provides a formal methodology for the development of bus scheduling models, which can be used by bus designers to improve designs that need to account for real-time traffic. The approach is to specifically account for non-ideal bus behavior that occurs in actual bus implementations. Applications can also be modeled, which allows the tool to be used to choose among different, specific bus scheduling policies or tune the bus for improved performance. Schedulability checks are performed by combining the application model with the bus model. If the application is not schedulable, the bus model can be changed, certain parameters in the database can be changed, or the application model can be changed. Similar tools have been developed for real-time/multimedia operating systems [Kettler *et al.*, 1995a] and networks [Sathaye, 1993].

The primary difference between CAISARTS and these tools lies in the scope and overall goal of CAISARTS. CAISARTS is designed to provide broad, text-based advice concerning allocation, schedulability and implementation, while these tools provide detailed analysis concerning a particular architectural component. CAISARTS can advise the user directly of competing alternatives; the tools discussed in this section can also provide analysis for different approaches, but only if the user configures the tool, one approach at a time, and invokes schedulability analysis. Conceptually, CAISARTS can be used in conjunction with these tools.

3 Overview of CAISARTS

CAISARTS is a modular set of functional components that communicate through a common underlying object management system [Goettge *et al.*, 1995]. The two primary functional components are the inference engine and the Graphical User Interface (GUI), which provides access to the objects that describe the environment as well as access to rules and rule sets. This section first discusses the GUI and the objects accessible by the GUI, and then gives an overview of the current rule set. This section concludes with a brief discussion of how CAISARTS uses these rules to generate analysis and advice for the real-time application designer.

Figure 1 shows the main window of CAISARTS. The window is organized into *containers*; a container is used to collect objects or other containers. The type of a container is either a *rule container*, which includes rules related to a particular topic, an *environment container*, which includes objects used to model the environment, or a *system approach container*, which is used to record the approach the user has taken (or is investigating) for the real-time application (such as "fixed priority scheduling with Rate Monotonic priorities"). As shown in Figure 1, **Knowledge Base** is the highest-level rule container, which contains twelve other rule containers. There are currently 137 rules defined, representing an average of about 11 rules per container. Rules will be discussed later in this section. The rest of the window displays environment containers and system approach containers. Shown in Figure 1 are: **Task Group Container**, which contains the tasks to be allocated and scheduled; **Schedulable Entity Container**, which contains objects used to implement the tasks; **Time Constraint Container**, which contains objects used to specify deadlines for tasks, schedulable entities, or task groups; **Hardware**, which contains objects both for the hardware architecture as well as the individual components; and **Sched Algorithm Container**, which contains objects that specify the scheduling policies to be used on the processors as well as the parameters associated with these policies. Not shown in Figure 1, but accessible by scrolling the window, are **POSIX Implementation Approach Container**,

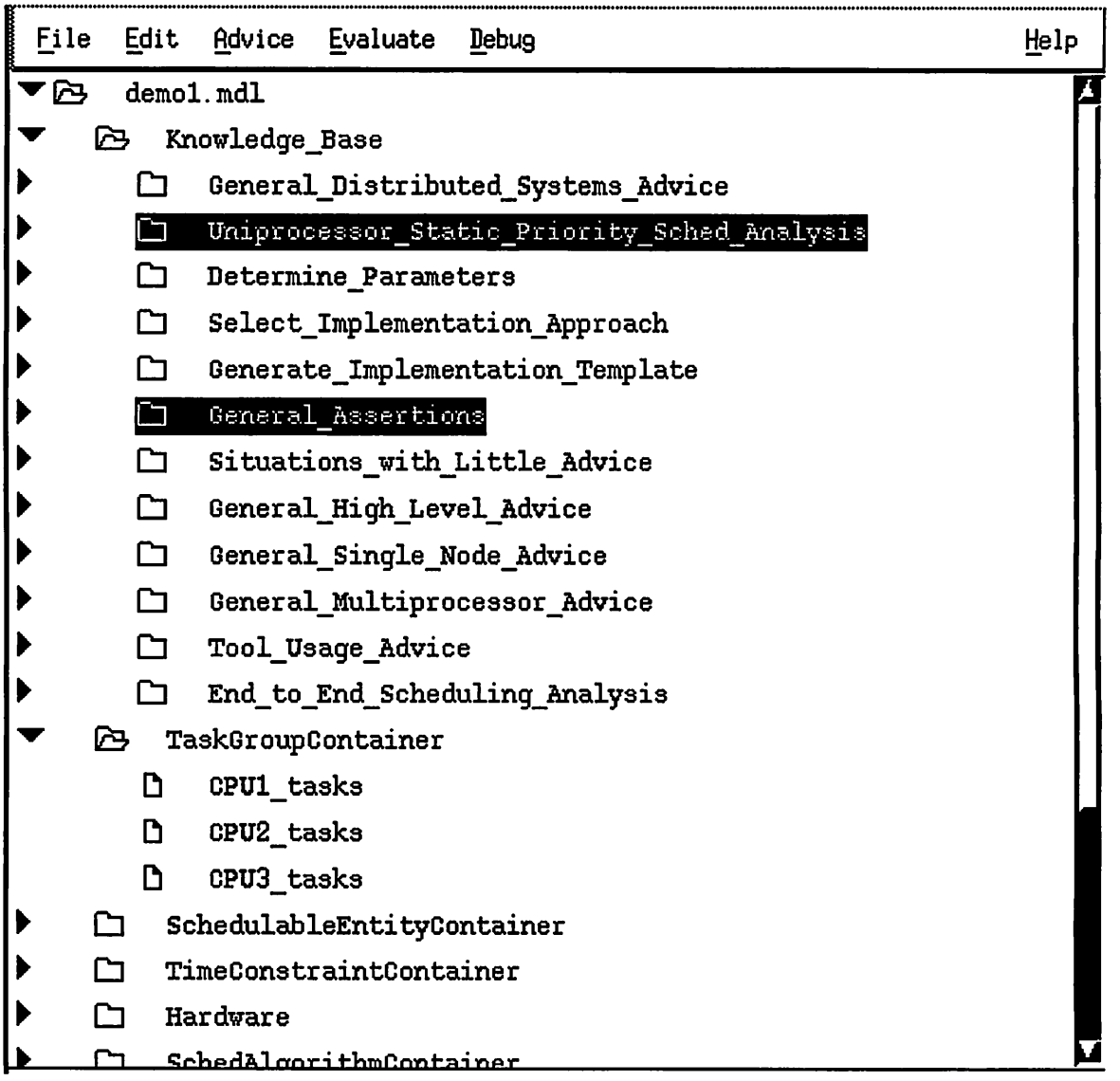

Figure 1: CAISARTS Main Window

which contains objects used to specify how POSIX [Gallmeister, 1995] services are used to implement the scheduling algorithms; **End to End Marker Container**, which contains objects used to logically connect a series of tasks or schedulable entities into a single entity; **Shared Resource Container**, which specify properties of individual objects accessible by schedulable entities or tasks; and **Shared Resource User Container**, which contains objects that specify the interactions of a task or schedulable entity with a shared resource; and **Communication Requirement Container**, which contains objects that specify a communication between tasks or schedulable entities.

The approach of the design of the objects that model the environment and the objects that capture the system approach is to make the objects robust and capable of modeling current real-time system research. The definitions of some of these objects are shown in Figure 2. Objects for Dynamic Priority Scheduling Algorithm, Table Driven Scheduling Algorithm,

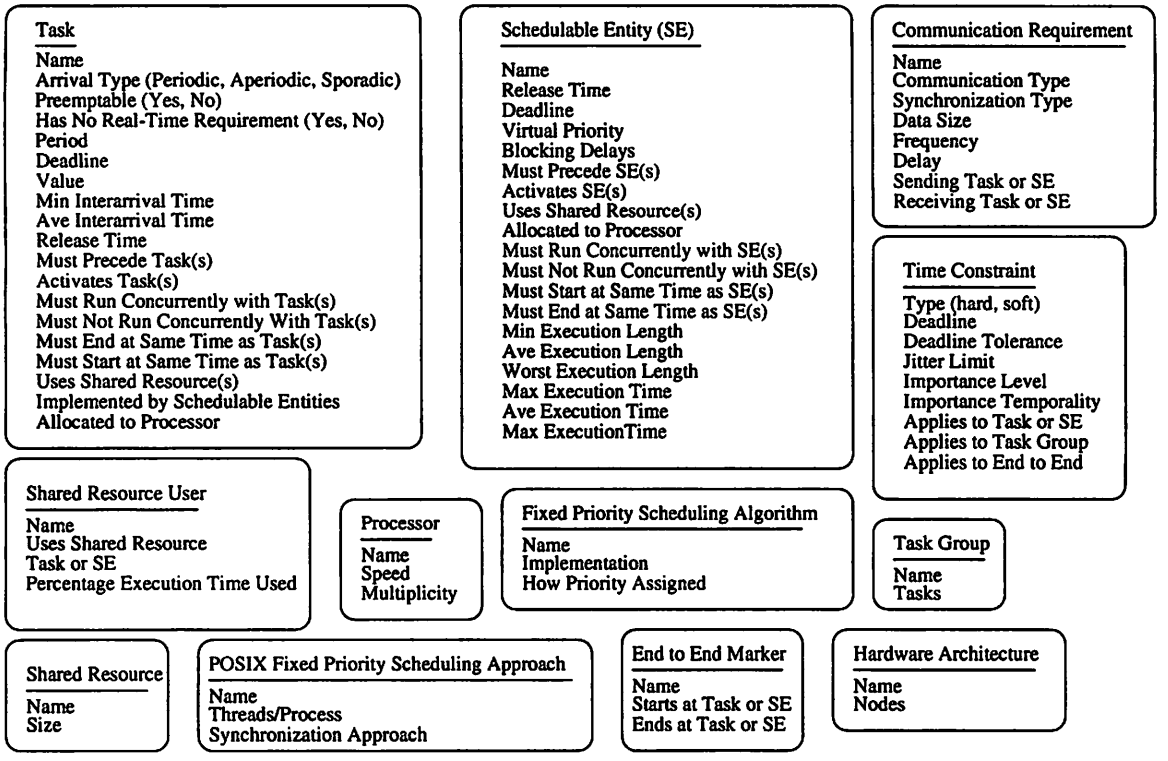

Figure 2: CAISARTS Environment Objects and System Approach Objects

and Dynamic Guarantee Scheduling Algorithm are similar to the Fixed Priority Scheduling Algorithm and have not been shown because of lack of space. In addition, POSIX Dynamic Priority Scheduling Approach, POSIX Table Driven Scheduling Approach, and POSIX Dynamic Guarantee Approach are similar to POSIX Fixed Priority Scheduling Approach and have been omitted. Most of the objects in Figure 2 do not require explanation; however, it should be noted that the conventional notion of task has been separated into two distinct objects: the *task* object captures many of the requirements of work to be performed, while the *schedulable entity* object captures work that can be dispatched by the operating system. The representation of task and schedulable entity as first-class data structures allow the modeling and analysis of complex multi-part implementations, such as a hardware interrupt and polling schedulable entity. It also allows the direct representation of a schedulable entity that services many tasks, as would be the case with a Sporadic Server used for many aperiodic tasks. These data structures reflect an early emphasis on objects that capture allocation and high-level task properties. A simple form of objects related more to low-level schedulability concerns—such as the objects for Fixed Priority Scheduling Algorithm and POSIX Fixed Priority Scheduling Algorithm—is adequate for the current version of CAISARTS, because references to implementation overheads are captured implicitly in rules, rather than explicitly as slots in objects. As CAISARTS develops, these objects will be revised to become more extensive, thus allowing the representation of more implementation details that affect scheduling.

The CLIPS system [Sof, 1994] provides the language by which to encode expert scheduling knowledge. A rule in CLIPS is a collection of conditions and the actions to take when the conditions are met (an IF-THEN rule). An important feature of CAISARTS is that individual

rules can be instantiated or modified by the GUI. Figure 3 shows the interface to an example rule that performs a schedulability bounds test based on fixed priority scheduling with Rate Monotonic priority assignment. The *Conditions* are being modified in this figure; other parts

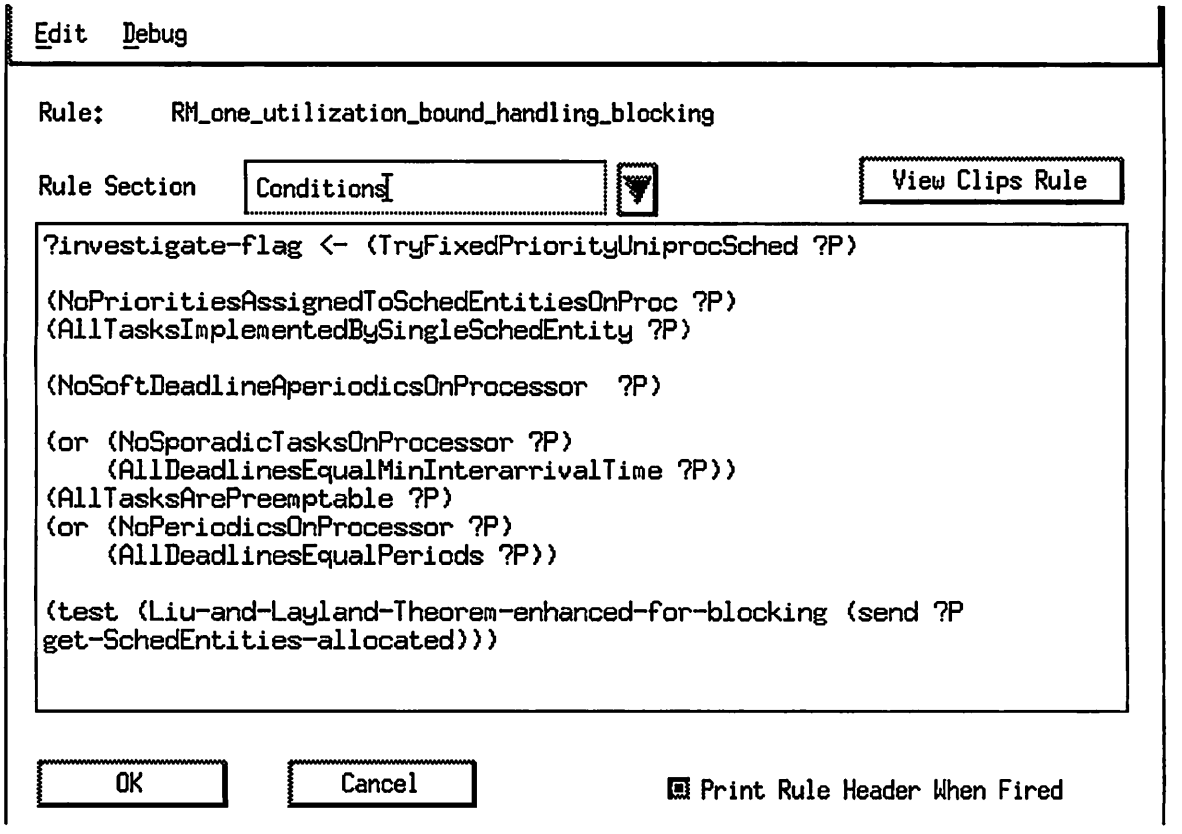

Figure 3: Example Rule

of the rule that can be modified include *Advice*, *Comments*, *PrePrintActions*, *PostPrintActions*, *Rationale*, and *Source*. Currently, rules must be encoded directly in the CLIPS language.

To date, we have developed general rules for uniprocessors and distributed systems. Rules and analysis for uniprocessors related to Rate Monotonic and sporadic server algorithms have been developed in greater depth than for other algorithms[1] primarily because a well developed theory exists for uniprocessor scheduling for Rate Monotonic. The state of the art in distributed systems is not very well developed, which means that very little precision can be incorporated into the associated rules. Instead, rules exist to provide the user with alternative algorithms and approaches to investigate further. For example, these rules assume that modules have been designed and that some initial hardware configuration has been chosen. The rules then attempt to provide options for further investigation for both allocation and distributed scheduling, as well as itemizing the ways in which these two levels interact. Examples of some of the rules are shown in Figure 4. Translating and encoding these rules is sometimes difficult; sometimes the "then" part can be encoded as text-based advice, while other times it can be encoded to trigger

[1] Allocation/scheduling approaches or algorithms currently described by or reflected in rules to a lesser degree in CAISARTS include 0-1 integer programming, simulated annealing, time windows, focussed addressing, bidding, the Spring complex task set allocation algorithm, EDF, Least Laxity, Xu/Parnas, etc.

the firing of other rules (see below, and Section 4).

IF the task set consists of periodics and aperiodics with soft deadlines
THEN consider the sporadic server algorithm

IF there is complete knowledge of deadlines, computation times, precedence constraints, future release times, and the external workload is well-behaved
THEN consider static scheduling

IF a static distributed real-time system and guarantees are required off-line
THEN consider simulated annealing, the Spring scheduling heuristic, or a window-based approach

IF Rate Monotonic is chosen
THEN add context switch overhead equal to 2 plus 2 times the maximum number of preemptions

IF using semaphores
THEN avoid unbounded priority inversion by using the priority ceiling protocol (PCP) or a planning-based algorithm

Figure 4: Example Rules in CAISARTS

The main window (Figure 1) is used to ask CAISARTS to generate expert advice. To do this, the user selects rule sets to be considered for "firing", and then uses the *Advice* option at the top of the window to engage the inference engine of CAISARTS. The user can also select individual rules by opening a rule set and selecting the appropriate rule (none of the twelve rule sets in Figure 1 are shown in the "open" state). In Figure 1, **General Assertions** and **Uniprocessor Static Priority Scheduling Analysis** are selected. The manner in which selected rule sets are used to generate advice is as follows. In CLIPS, rules execute (or fire) based on the existence of facts. In CAISARTS, facts include both the initial facts, which are the instantiated environment objects and system approach objects, and assertions made as rules fire, which are in general conclusions that are derived from the instantiated environment objects and system approach objects. An example of a simple derived fact is "NoPeriodicsOnProcessor CPU1". The sole purpose of the **General Assertions** rule container is to collect rules that only assert facts, and do not provide text-based advice for the user. These facts are usually needed for the rules that provide analysis and/or advice, so **General Assertions** is almost always selected. The way a program—the selected rules and the initial facts—executes is by continued application of the following cycle:

1. The conditions of the selected rules are matched against the current facts; those that have all of their conditions satisfied are deemed capable of being fired.
2. One of the rules capable of being fired is chosen for firing. The way in which this choice is made can be complicated and is beyond the scope of this paper.
3. The chosen rule is fired, which means that its actions are executed; performing the actions in the rule may cause the addition of new facts, the modification or removal of existing facts, or the printing of text-based advice to a window.

This cycle is repeated until there are rules capable of being fired, as defined in step 1. Depending on the style of the rule, the actual analysis can occur in the conditions part of the rule or the actions part of the rule.

4 Rule Base

The majority of this section discusses issues that arose as the rule base was designed and implemented. It is important to observe that research into real-time systems is extensive, yet piecemeal. Analyzable scheduling approaches (a viable approach may be a collection of algorithms) that are comprehensive and integrated are lacking. CAISARTS has been designed to represent knowledge related to scheduling of real-time systems, accepting that this knowledge is not uniformly deep. The scope of the rule base is ultimately to handle: preemptable and non-preemptable tasks, periodic and non-periodic tasks, tasks with multiple levels of importance (or a value function), groups of tasks with a single deadline, end-to-end timing constraints, precedence constraints, communication requirements, resource requirements, placement constraints, fault tolerance needs, tight and loose deadlines, and normal and overload conditions. In addition, CAISARTS will address the interfaces between CPU scheduling and resource allocation, I/O scheduling and CPU scheduling, CPU scheduling and real-time communication scheduling, and local and distributed scheduling. For now, comprehensive rules do not exist in the real-time community and will probably not exist for many years. The approach taken in CAISARTS is to codify real-time systems knowledge in whatever form and precision is known. As more precise and comprehensive results are obtained, existing rules can either be removed or modified to increase their precision.

4.1 Types of Rules

To date, the emphasis has been to develop the structure of the rule base, rather than attempting to make the rule set as comprehensive as possible. Twelve rule sets have been defined, which are shown in Figure 1 and described in Table 1. The number and breadth of rule sets will expand in the future.

4.2 Issues in the Design and Coding of Rules

This section describes issues that arose when designing and implementing the rule base.

Vagueness of rules. Real-time system scheduling knowledge is often vague and heuristic in nature. An example of such a rule is "If tasks communicate a lot, then attempt to cluster them on the same node." The problem with this type of rule is encoding this knowledge such that the rule fires at appropriate times. Either of two approaches can be taken, each with negative implications:

1. Attempt to quantify "a lot", which can be quite difficult.
2. Make the conditions under which this rule will fire loose (e.g. any time two tasks communicate at all), and explicitly leave the judgment of "a lot" up to the user (e.g.

Category	*Description*
General Assertions	assert internal facts used to fire other rules (rather than provide textual advice)
General High Level Advice	design methodology advice, such as high-level requirements and task implementation approaches
General Distributed Systems Advice	includes allocation advice and general advice related to distributed systems
General Single Node Advice	approaches particular to uniprocessors
General Multiprocessor Advice	approaches particular to multiprocessors
Uniproc. Static Priority Sched. Analysis	includes rate monotonic, deadline monotonic, and arbitrary priority assignment analysis
End-to-End Scheduling Analysis	includes Distributed Rate Monotonic and Holistic Scheduling approach
Determine Parameters	advises how to configure certain approaches after they are selected by the user
Select Implementation Approach	focuses on POSIX; advises appropriateness of certain POSIX constructs
Generate Implementation Template	provides code templates
Situations with Little Advice	advises that certain knowledge has either not been implemented yet in CAISARTS or state-of-the-art is lacking
Tool Usage Advice	checks consistency of objects input by the user; advises user of likely human error in data input

Table 1: Existing Rule Set Containers

the advice is roughly "if you, the user, feel that the two tasks communicate a lot, then place them on the same node, if possible.") The obvious problem with this approach is that the purpose of the tool–which is to encode expert real-time system advice related to scheduling and advise a user at appropriate times of this knowledge—is perhaps defeated by leaving the user to make expert-level decisions.

There is not an easy solution to this problem. To date, the appropriateness of each approach has been evaluated on a per-rule basis.

Form of rules. Some expert analysis can be rather computation-intensive and time consuming. There is some question how to write these rules: should they perform the analysis "in the background", or should they state a step-by-step procedure that the user can take in order to perform the analysis? If CAISARTS is to perform the analysis in the background, it could require the temporary instantiation of objects, firing of rules, and then retraction of objects. This results in slower feedback from CAISARTS to the user—analysis and advice that may not even be directly required by the user. In general, the user should be given some choice concerning the amount of analysis performed by the normal rule-firing procedures. We have chosen to encode computationally-intensive rules as step-by-step procedures that the user can follow. This greatly improves the efficiency of the firing of the rules.

Inconsistency of rules. There is a difference between inconsistent beliefs among sets of researchers and inconsistent rules derived from knowledge that is inherently consistent with

each other. A strength of CAISARTS is that it can represent inconsistent beliefs between research communities, and in fact present them to the user, explicitly noting that they are two differing sets of opinions. However, if inconsistent rules are added (one rule is added without the knowledge of the other rule), then the user could get advised inconsistently, without either the system or the user knowing. Both a positive *and* a negative aspect of a rule-base system is that the rules are relatively independent from each other. Rules can be added freely, but it is easy to introduce inconsistencies, especially considering the nature of some of the rules (they can be vague). To date, there has been careful control of this problem, but it is conceivable that this potential problem could be exacerbated as the tool is released to users.

Level of advice. CAISARTS is intended to provide advice related to scheduling at all stages of design and implementation of a real-time system. Ideally, CAISARTS should provide only the most appropriate advice at only the most appropriate time. The problem is that this ideal amount of advice is related to the "sophistication" of the user—e.g. an designer only interested in the exact characterization form of deadline monotonic analysis on a particular node should not be given advice concerning how to allocate tasks. The approach to this problem in CAISARTS has been partially addressed by enabling the user to select rule sets and even individual rules, but this does not allow CAISARTS to completely infer the level desired by the user. Even within a single rule, it is difficult at times to determine when a rule should fire, and when a rule should not fire. For example, assume that the user has selected a fixed priority scheduling algorithm and assigned priorities such that all tasks make their deadlines in the worst-case. If this priority assignment is non-optimal, should the user be informed? While the current priority assignment is valid for the current task set properties, it may not be the "best" priority assignment in terms of worst-case response time or resistance to minor parameter modifications. In this sense, although this advice is not directly required by the user, providing this advice at this point could facilitate a more robust design.

5 Evaluation/Validation

As part of the validation of the tool, CAISARTS was applied to the Submarine Passive Sonar scenario described by Molini, Maimon, and Watson [Molini *et al.*, 1990]. This specification and implementation of a Submarine Passive Sonar application was chosen both because it adequately describes a real-time environment in some detail, and because it presents a sample *implementation* of a real-time distributed system to perform the Submarine Passive Sonar task. The application is complex: there are three nodes with communication across nodes, and there are both aperiodic and periodic tasks. In this section, after describing the environment and application design in more detail, CAISARTS is shown to provide analysis that is consistent with the analysis contained in [Molini *et al.*, 1990], under the same simplifying assumptions—that nodes are independent, and the aperiodics are not included in the analysis. To further illustrate the capabilities of the tool, CAISARTS is shown to extend the schedulability analysis of [Molini *et al.*, 1990] by performing a schedulability analysis of the sporadic tasks. This also illustrates implementation advice that is provided by CAISARTS that does not appear in [Molini *et al.*, 1990]. Next, three end-to-end constraints across nodes are added and analyzed, illustrating the ability of CAISARTS to provide advice for a more complex, distributed, real-time system.

Again, this represents analysis techniques beyond those in [Molini *et al.*, 1990]. We will not show the step-by-step trace of how to use CAISARTS on the Submarine Passive Sonar environment; instead, the presentation that follows summarizes the output of CAISARTS at various stages of its use on that environment.

5.1 Submarine Passive Sonar Domain

Submarine Passive Sonar is used to detect the presence of objects in the sea based on the sounds emanating from the objects themselves. Signals received from hydrophones placed in the water are digitized and processed to form a number of beams. Beams are then processed further to form detected objects, which are analyzed and then tracked. Detected objects are also used to steer additional signal processing.

There are eight categories of time-critical functions for Submarine Passive Sonar: signal conditioning, beamforming, detection, tracking, analysis and classification, stabilization, time synchronization, and audio. The particular implementation described in [Molini *et al.*, 1990] consists of a total of 28 tasks. The signal conditioning and the beamforming functions are performed in custom-built hardware (4 tasks), leaving the remaining functions (24 tasks) to be executed on standard hardware. Twenty-two of the tasks are periodic, and 2 tasks are aperiodic with hard deadlines (*sporadic* tasks). The CPU, period, deadline, and worst-case execution time for each of the periodic tasks are shown in Table 2.

Task	*CPU*	*Period (ms)*	*Deadline (ms)*	*WCET (ms)*
Update Steering	1	100	100	22.01
Estimate Tracks	1	250	250	5.78
Recompute Delays	1	250	250	5.78
Format Display	1	250	250	114.65
Rebuild Bound	1	250	250	2.75
Adjust Clock	1	125	125	2.75
Receive New Fix	1	50	50	1.83
Get Cursor	2	100	100	11.00
Update Cursor For Tracks	2	100	100	9.17
Update Cursor for Detects	2	100	100	14.67
Display Comparison	2	100	100	4.59
Automatic Comparison	2	250	250	57.32
Estimate Tracks	2	250	250	5.78
Show Det Display	2	100	100	7.34
Show Track Display	2	100	100	4.59
Adjust Clock	2	125	125	2.75
Receive New Fix	2	50	50	1.83
Forward Tracks	3	100	100	4.59
Build Time Message	3	125	125	2.75
Adjust Clock	3	125	125	2.75
Compute Attitude	3	50	50	9.17
Receive New Fix	3	50	50	1.83

Table 2: Periodic Tasks in Passive Sonar Implementation

5.2 Independent Nodes without Sporadic Tasks

The GUI of CAISARTS is used to input the nodes, tasks, and schedulable entities, with the same allocation as in Table 2. At this point, the aperiodic tasks have not been input into CAISARTS. Rules for analysis assuming independent nodes are invoked; the analysis rules in CAISARTS indicate that Static Priority scheduling is appropriate for each of the nodes. The schedulability of CPU3 is determined by a bounds test, while exact characterizations are used for CPU1 and CPU2. A partial listing of the advice for CPU1 and CPU3 is shown below:

```
ADVICE: Select Static Priority scheduling for Processor CPU1, and
  assign priorities according to the Deadline Monotonic policy.
RATIONALE: Every task will be guaranteed to complete before its
  deadline:

   Task                        Worst case completion      Deadline
                                    time (sec)              (sec)
   RecomputeDelays                   0.032                 0.250
   FormatDisplay                     0.177                 0.250
   RebuildSound                      0.180                 0.250
   AdjustClock1                      0.027                 0.125
   ReceiveNewFix1                    0.002                 0.050
   UpdateSteering                    0.024                 0.100
   EstimateTracks1                   0.186                 0.250
SOURCE: Klein et al. A Practitioner's Handbook for Real-Time
  Analysis: Guide to Rate Monotonic Analysis for Real-Time
  Systems. Kluwer Academic Publishers. Boston, MA. 1993. p.4-27.
 See also:
    Audsley, Burns, Richardson, and Wellings. Hard Real-Time
     Scheduling: The Deadline Monotonic Approach. Proceedings of
     the 8th IEEE Workshop on Real-Time Operating Systems and
     Software. 1992.
    Joseph and Pandya. Finding Response Times in a Real-Time
     System. The Computer Journal (British Computing Society),
     29(5), Oct 1986, pp. 390-395.

ADVICE: Select Static Priority scheduling for Processor CPU3, and
  assign priorities according to the Rate Monotonic policy.
RATIONALE: Every task will be guaranteed to complete before its
  deadline. Actual utilization (0.310) plus blocking (0.0) is less
  than the theoretical maximum in the worst case for 5 tasks, which
  is 0.743.
SOURCE: Klein et al. A Practitioner's Handbook for Real-Time
  Analysis: Guide to Rate Monotonic Analysis for Real-Time
  Systems. Kluwer Academic Publishers. Boston, MA. 1993. p.4-10.
 See also:
    Liu and Layland. Scheduling Algorithms for Multi-Programming in
     a Hard Real-Time Environment. Journal of the ACM, 20(1), Jan
     1973, pp.40-61.
    Sha, Rajkumar, and Lehoczky. Priority Inheritance Protocols: An
     Approach to Real-Time Synchronization. IEEE Trans. on Computers,
     39(9), Sept 1990, pp. 1175-1185.
```

These are the basic results given in [Molini *et al.*, 1990]—that static priority scheduling is appropriate. In the next part, we extend the analysis in that paper by adding the two sporadic

tasks according to the allocation scheme of [Molini *et al.*, 1990] and illustrate the analysis of end-to-end scheduling requirements.

5.3 Analysis and Implementation of Sporadic Tasks

Task	*CPU*	*Min Interarrival Time (ms)*	*Deadline (ms)*	*WCET (ms)*
Specific Comparison	1	200	200	9.17
Comparison Request	2	100	100	4.59

Table 3: Sporadic Tasks in Passive Sonar Implementation

The two sporadic tasks described in Table 3 are instantiated and allocated. CAISARTS is used to determine the approach taken to implement the sporadic tasks (i.e., the mapping of task to schedulable entity), so rule sets will be selected and fired after instantiating the two tasks, but before specifying the schedulable entities for the two tasks. CAISARTS generates the following advice concerning how implementation and schedulability interact. Only advice for *Comparison Request* is shown; advice for *Specific Comparison* is similar.

```
ADVICE: A sporadic task can be implemented in a variety of ways;
  the best way depends on the particular semantics of the situation:
  -- as a single, hardware-priority interrupt modeled as a single
     schedulable entity
  -- as a HW interrupt and a software interrupt service routine,
     modeled as two separate schedulable entities
  -- as a single, polling task, modeled as a single schedulable
     entity
  -- as a HW interrupt and a Sporadic Server, modeled as two separate
     schedulable entities.
There are Pros and Cons for each situation. For more details,
consult page 5-61 of the Handbook of Rate Monotonic Analysis.

ADVICE: To implement ComparisonRequest as a hardware interrupt:
  [1] Instantiate a single schedulable entity, and allocate it to
      Processor CPU2. Fill in the worst-case execution time of
      the entire response.
  [2] Assign 11 as the priority of the new schedulable entity.
      This is equivalent, for this schedulable entity set, to a
      hardware interrupt.
  [3] Make the task (ComparisonRequest) point at the new
      schedulable entity through its 'ImplementedBy' slot.
  [4] Re-invoke the uniprocessor static priority scheduling rules to
      determine the schedulability of Processor CPU2 as a whole.

ADVICE: To implement ComparisonRequest with a sporadic server:
    [1] Instantiate two schedulable entities, and allocate them to
        Processor CPU2.
         -- the first schedulable entity is the hardware interrupt.
            Assign a worst-case execution time that is the amount
            of time needed to service the interrupt.
         -- the second schedulable entity is the sporadic server. If
            schedulability analysis shows that this is appropriate, then
```

```
        when it comes to coding the sporadic server, the sporadic
        server replenishment period will be 100 milliseconds,
        and the execution capacity will be the worst-case execution
        time of the second schedulable entity, which you also have
        to specify.
    [2] Make the first schedulable entity point at the second schedulable
        schedulable entity through its 'MustPrecede' slot.
    [3] Make the task (ComparisonRequest) point at the FIRST
        schedulable entity through its 'ImplementedBy' slot.
    [4] Re-invoke the uniprocessor static priority scheduling rules to
        the schedulability of Processor CPU2 as a whole.
```

The Sporadic Server approach is chosen, and reinvoking the schedulability test indicates that timing constraints will be met (the time needed to service the interrupt is assumed to be 0.5 milliseconds). The results of these tests are not shown because they are very similar to results shown in the previous section.

We next specify the implementation approach of the nodes. Objects that specify a fixed priority scheduling approach are instantiated for each node, as well as objects that specify that a POSIX approach will be pursued. We have looked into POSIX to determine the areas related to scheduling where advice may be developed. In particular, we have a first collection of rules that address, albeit in a limited manner, the following areas: semaphores, mutexes, condition variables, signals, timed locks, timer functions, memory management, and I/O. We have not yet considered IPC, shared memory objects, and files. Some of that advice that is specific to POSIX that is generated by CAISARTS is shown below.

```
ADVICE: The periodic tasks on CPU2 should <NOT> be
  be coded with alarm(), pause(), SIGALRM, and sleep().
RATIONALE: Conceptually, these functions are easy to use and
  understand. Alarm() sets the period, pause() suspends the process
  or thread after a 'completion' in one period and a SIGALRM resumes
  the processor thread. Sleep() suspends the process or thread for a
  specified interval. However, the implementation of these primitives
  does not give any guarantees on the latencies of when processes
  will be active again.

ADVICE: One way in which to implement the periodic tasks on CPU2
  is to use the following timer functions with absolute intervals:
      timer_create(), timer_delete(), timer_settime(), timer_gettime(),
      timer_getoverrun()
RATIONALE: you need to create a timer, determine when it will be
  'activated', and set the reload value for the time. It is recommended
  to use the absolute value reload.

ADVICE: Implement the Rate Monotonic scheduling on Processor CPU2
  by using the POSIX FIFO.
RATIONALE: Only RR and FIFO are 'standard' for POSIX.
```

Additional detailed implementation advice is generated when the user specifies, for example, whether a thread or a process model will be pursued (advice for choosing between the two is also included in CAISARTS), and which synchronization protocol will be selected. This advice is not shown here because of lack of space.

A particular strength of the tool is that CAISARTS provides a framework for making the user aware of the effect of real-world overheads on schedulability. Consideration of OS

implementation costs on the schedulability of a task set is complex, but must be performed before any confidence can be attained in the system. For example, the designer must account for processing overhead for new events such as clock ticks, messages arriving, activating tasks at periods, I/O ready signals, interrupts from sensors, etc. In a future version of CAISARTS, the user will be allowed to directly represent overheads of particular operating systems. Once the user selects a particular implementation approach and inputs the overheads, CAISARTS can re-perform schedulability analysis. Detailed specifications of the operating system as well as details of the implementation approach also allows CAISARTS to generate code templates. Code generation is not shown here because it is currently under development.

5.4 End-to-end Schedulability

To further validate CAISARTS the set of tasks and semantics of the application of [Molini *et al.*, 1990] was extended to include the three end-to-end constraints shown in Figure 5. The

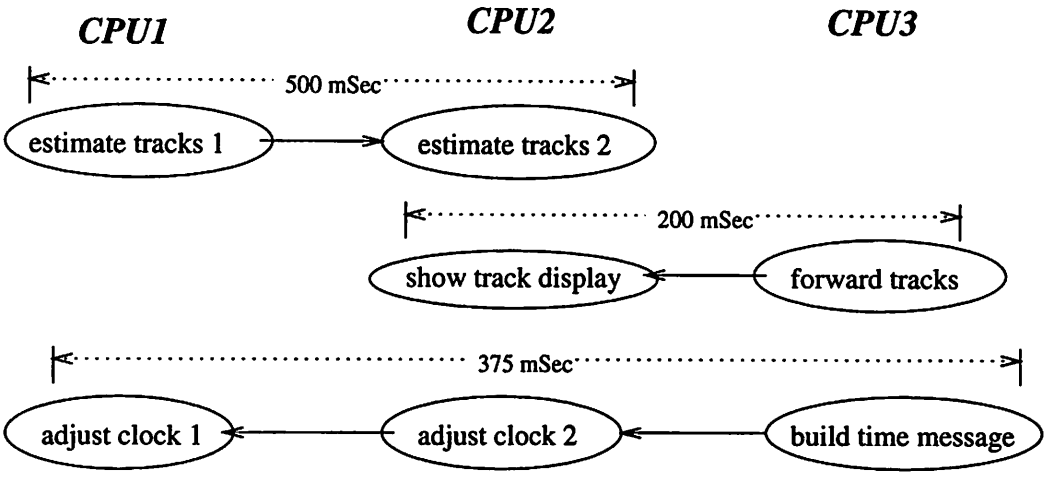

Figure 5: End to End Constraints

description of the first end-to-end constraint, *EstimateTracksEE*, is that the task *EstimateTracks1* now must send a message to *EstimateTracks2* before *EstimateTracks2* should execute. In addition, there is an end-to-end deadline of 500 milliseconds. In order to simplify this example, the network is modeled as a point-to-point medium. In other words, the worst-case transmission time can be bounded, and no explicit scheduling of the network is necessary. The worst-case transmission delay is assumed to be 5 milliseconds.

Selecting and invoking rules for end-to-end analysis produces the following advice. Only the advice pertaining to *EstimateTracksEE* is shown.

```
ADVICE: Be aware that the Handbook of RMA includes a section for
  end-to-end analysis. In that model, the first task in an end-to-end
  response can be thought of as periodic, and every other task in
  the end-to-end response can be considered aperiodic. Currently,
  CAISARTS does not include rules for that advice. Consult the
  Handbook of RMA, p. 6-100.

ADVICE: One way to attempt to guarantee the end-to-end deadline on
```

```
  EstimateTracksEE is to attempt the Holistic Approach. In this
  approach, the network delay can be accounted for in the sending
  task(s). Because the sum of the periods of the tasks involved in
  the end-to-end constraint is less than or equal to the end-to-end
  deadline, this approach can be pursued.
SOURCE: Tindell and Clark. Holistic Schedulability Analysis for
  Distributed Hard Real-Time Systems. Microprocessing and
  Microprogramming, 40(1994), pp. 117-134.

ADVICE: To use the holistic approach to the end-to-end constraint
  EstimateTracksEE,
    [1] Reduce the deadline of Task EstimateTracks1 5 milliseconds.
    [2] Perform a static-priority scheduling analysis on the
        system. Pay particular attention to those processors (CPU1)
        that have their task set attributed modified as a result of
        the previous steps.
RATIONALE: If the tasks on the processor on which the sending task
  is allocated are still schedulable after the deadline of the
  sending task is reduced, then the end-to-end timing constraint will
  be met.
```

As can be seen from this advice, the focus of scheduling knowledge for distributed end-to-end constraints has been the Holistic Approach [Tindell and Clark, 1994] and the Distributed Rate Monotonic approach [Klein *et al.*, 1993]. In addition, distributed rate monotonic is only acknowledged as an option for the user to investigate. A future improvement planned for CAISARTS is to replace this temporary, "placeholder" rule with a set of rules that provide in-depth analysis and advice pertaining to distributed rate monotonic.

To continue with the use of CAISARTS on the end-to-end constraints, the Holistic Approach is taken for all three end-to-end constraints, as described in the advice generated by CAISARTS. Schedulability tests are re-executed, which show that the Holistic Approach is appropriate for all three end-to-end constraints, because all tasks will complete by their deadlines. For example, the schedulability analysis produced by CAISARTS for CPU1 is shown below:

```
ADVICE: Select Static Priority scheduling for Processor CPU1, and
  assign priorities according to the Deadline Monotonic policy.
RATIONALE: Every task will be guaranteed to complete before its
  deadline:
```

Task	Worst case completion time (sec)	Deadline (sec)
RecomputeDelays	0.038	0.250
FormatDisplay	0.183	0.250
RebuildSound	0.186	0.250
AdjustClock1	0.027	0.125
ReceiveNewFix1	0.002	0.050
UpdateSteering	0.024	0.100
EstimateTracks1	0.032	0.245

6 Conclusions

As an aid in addressing the complexity of scheduling issues in real-time system design, we have implemented a multi-level real-time system design assistant that contains rules, advice,

explanations, analysis, and code templates. While some real-time scheduling analysis tools exist, they are typically limited to specific algorithms, to uniprocessors, or are essentially only simulators. CAISARTS is more comprehensive in that it is a tool that addresses (i) uniprocessors and distributed systems, (ii) algorithmic, analysis and implementation issues, and (iii) can provide code templates.

We have demonstrated that real-time scheduling knowledge can be codified at various levels of detail: the conceptual level, the analysis level, and the implementation level. CAISARTS is able to provide advice in choosing a good scheduling algorithm, prevent errors due to subtleties, anomalies, and assumptions, provide partial, or in some simple cases, even complete analysis, and provide advice and code for implementation. Increased productivity, fewer errors, and increased understanding of the system via rationale and explanations are the projected outcomes.

We have completed several experiments of using the tool; an extended example is contained in this paper for the passive sonar application reported in [Molini *et al.*, 1990]. In these experiments, the tool's advice was validated by comparing to that implementation and was able to provide solutions beyond what was presented in [Molini *et al.*, 1990] both for sporadic tasks and end-to-end constraints.

However, since the scheduling field is so complex (there is an infinite variety of algorithms and situations) and much research still needs to be done, for many problems only guidance and suggestions may be forthcoming rather than complete solutions. Similarly, we cannot hope to provide advice on all analysis techniques or implementations. Even in a limited domain there is not necessarily a single best algorithm. As understanding is accumulated, CAISARTS can continually be refined to address more application situations, algorithms, analysis techniques and implementations.

The current status of CAISARTS is that it is usable and can be demonstrated. While the current tool can handle a wide variety of system requirements, in the future we will continue to enhance the comprehensiveness of the tool by adding more depth to the algorithms currently supported as well as adding new algorithms. Plans for the immediate future include implementing rules that have been designed and written to provide scheduling advice related to fault tolerance.

Acknowledgments

CAISARTS has been jointly developed by Advanced System Technologies, Inc. and members of the Real-Time Systems Lab at the University of Massachusetts. The authors wish to thank Eric Brehm, Scott Anderson, and Krithi Ramamritham for important and constructive comments on earlier drafts of this paper.

References

[Gallmeister, 1995] B.O. Gallmeister. *POSIX.4: Programming for the Real World.* O'Reilly & Associates, Inc., Sebastopol, CA, 1995.

[Goettge *et al.*, 1995] R.T. Goettge, E. Brehm, C. Palczak, J.A. Stankovic, and M. Humphrey. Knowledge-based assistance for real-time systems. In *Proceedings of the 1st IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'95)*, Ft. Lauderdale, Florida, November 1995.

[Graham, 1969] R.L. Graham. Bounds on multiprocessor timing anomalies. *SIAM Journal on Applied Mathematics*, 17, 1969.

[Int, 1994] Introspect Technologies, Inc., Colorado Springs, CO. *iRAT Technical Overview*, 1994.

[Kettler *et al.*, 1995a] K.A. Kettler, D.I. Katcher, and J.K. Strosnider. A modeling methodology for real-time/multimedia operating systems. In *Proceedings of the 1st Real-Time Technology and Applications Symposium*, Chicago, Illinois, May 1995.

[Kettler *et al.*, 1995b] K.A. Kettler, J.P. Lehoczky, and J.K. Strosnider. Modeling bus scheduling policies for real-time systems. In *Proceedings of the 16th Real-Time Systems Symposium*, pages 242–253, Pisa, Italy, December 1995.

[Klein *et al.*, 1993] M.H. Klein, T. Ralya, B. Pollak, R. Obenza, and M.G. Harbour. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems.* Kluwer Academic Publishers, Boston, MA, 1993.

[Liu *et al.*, 1993] J.W.S. Liu, J.L. Redondo, Z. Deng, T.S. Tia, R. Bettati, A. Silberman, M. Storch, R. Ha, and W.K. Shih. PERTS: A prototyping environment for real-time systems. Department of Computer Science Technical Report UIUCDCS–R–93–1802, University of Illinois at Urbana-Champaign, May 1993.

[Molini *et al.*, 1990] J.J. Molini, S.K. Maimon, and P.H. Watson. Real-time system scenarios. In *Proceedings of the 11th Real-time Systems Symposium*, December 1990.

[Sathaye, 1993] S. Sathaye. *Scheduling Real-Time Traffic on Packet Switched Networks.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, May 1993.

[Sof, 1994] Software Technology Branch, NASA Johnson Space Center. *CLIPS Reference Manual, version 6.0*, January 1994.

[Tindell and Clark, 1994] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40:117–134, 1994.

[Tokuda and Kotera, 1988] H. Tokuda and M. Kotera. A real-time tool set for the ARTS kernel. In *Proceedings of the 9th IEEE Real-Time Systems Symposium*, December 1988.