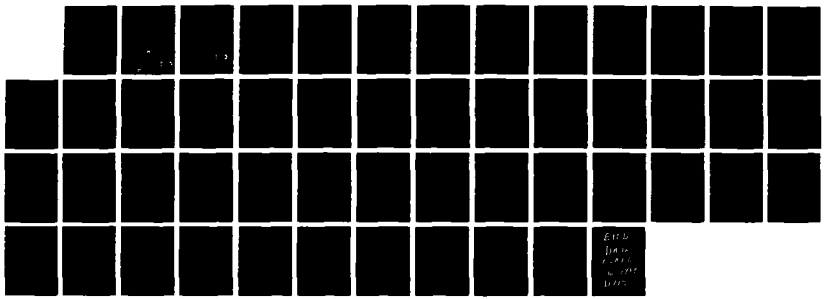
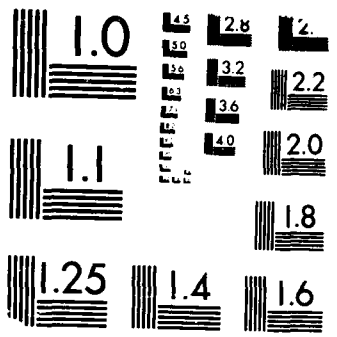
AD-A192 539 TEMPORAL RELATIONS AND STRUCTURES IN REAL-TIME OPERATING SYSTEMS(U) MARYLAND UNIV COLLEGE PARK INST FOR ADVANCED COMPUTER STUDIES.. S LEVI ET AL. DEC 87 1/1

UNCLASSIFIED UMIACS-TR-87-61 N00014-87-K-0241 F/G 12/7 NL





MICROCOPY RESOLUTION TEST CHART

UMIACS-TR-87-61
CS-TR-1954

December, 1987

Temporal Relations and Structures in Real-Time Operating Systems†

Shem-Tov Levi
Systems Design and Analysis Group
Department of Computer Science

Ashok K. Agrawala‡
Department of Computer Science and
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742

COMPUTER SCIENCE TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

UMIACS–TR–87–61 ✓
CS–TR–1954

December, 1987

Temporal Relations and Structures in Real–Time Operating Systems†

Shem–Tov Levi
Systems Design and Analysis Group
Department of Computer Science

Ashok K. Agrawala‡
Department of Computer Science and
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742

ABSTRACT

The temporal properties of objects in a real–time, distributed, fault–tolerant , reactive operating systems are defined and analyzed. Accordingly, properties associated with the schedulability of accepted jobs whose deadlines are guaranteed are examined. Special mechanisms that support temporal inference are proposed. These mechanisms support explicit time expression, precedence relations, and projections of the knowledge of real–time at different localities. Special data structures, called *calendars*, are proposed for management and planning of activities and for scheduling. Algorithms that verify schedulability of arriving requests are introduced, ensuring that already–given guarantees for already–accepted jobs are not violated.

† This work is supported in part by contract N00014–87–K–0241 from the Office of Naval Research to the Department of Computer Science, University of Maryland

‡ The authors are indebted to Satish K. Tripathi for the help and the constructive criticism he has provided throughout the preparation of this paper.

Contents

1 Introduction

In this paper we examine some aspects of the use of objects as the building blocks for constructing a real-time reactive system. A reactive system repeatedly responds to requests from its environment by producing outputs ([8]). In our case the requests are invocations of executable objects. The temporal properties that the objects in the system demonstrate, characterize the whole system's temporal and functional behavior. Therefore, when designing a real-time, distributed and fault-tolerant reactive operating system, the temporal properties of its objects are of crucial importance.

One major property in the temporal behavior of a hard real-time operating system is the reliability of the guarantee it provides for satisfying time constraints of jobs it has already accepted. This guarantee is the focus of this paper. A guarantee of this kind depends on the way in which new jobs are accepted and on verification of the schedulability of a new job before accepting it. In this paper we propose mechanisms that support temporal inference requirements for schedulability verification, along with algorithms that carry out the verification.

The paper is organized as follows. In the remaining of this section we outline the object architecture concept, the meaning of a deadline guarantee in a real-time system, and the participation of the time-service in the system management scheme. The following section provides notations for descriptions of temporal properties and relations. The proceeding section describes the concept of *calendars* as means of resource allocation management, object sharing, and deadline guarantee. The next section examines various properties of an object's calendar. The paper closes with some concluding remarks.

1.1 Objects Architecture: A Review of Previous Work

In [15,16] we have introduced an architecture for designing hard[1] real-time operating systems. This architecture is based on the use of highly encapsulated entities, called objects. An *object* is a distinct and selectively accessible software element that resides on one of the storage resources of the system. The *objects architecture* defines the objects as the elements that constitute the system. It also defines their classification, the relationships between them, the set of operations they are subjected to and execution parameters that permit scheduling them for execution and access.

Many software engineering research efforts have been invested in the use of objects to establish a concurrent programming development environment. In this context an object is captured as an entity whose behavior is characterized by the operations it is subjected

[1] Hard real-time systems are characterised by their property of having a nonrecoverable fault when a computation is not complete before its deadline.

to and the operations it carries out on other objects. The external view of an object (these operations) is its specification, and the internal view of an object is its implementation. We have previously introduced a point of view which is slightly different. There, we have considered the use of objects architecture in a system context, thereby expanding the above object definitions to describe elements and entities in a more general way. Yet, some of the properties that characterize objects in software development context ([3]) are valid in the system architecture context as well, in that:

- An object has a state.
- There is a set of actions to which an object is subjected and a set of actions it requires from other objects.
- It is denoted by a name.
- It has restricted visibility of (as well as by) other objects.

We have shown how our object-oriented system design methodology provides means to construct systems with a high degree of deterministic and predictable timing properties. This determinism, together with the required fault tolerance schemes, set up a time constraint oriented system. In the above papers we have defined a classification of object types, the set of operations each of the object types is associated with, and their relationships. A conceptual model has been considered in our analysis of the applicability of objects architecture for a real-time distributed fault tolerant operating system. Issues of creation, deletion and access for manipulation and state verification, have lead us to define the *joint* that contains the following parts:

- A context independent pointer for the naming network, to allow a multi-user, selective sharing of the object.
- An owner/user justification structure.
- Resource requirements.
- A ticket check mechanism for the protection scheme.
- A time constraint for an executable object.
- A replica/alternative control mechanism for the fault tolerance scheme.

In our model, objects that relate to each other, or in other words objects that satisfy a given semantic link, are connected via the owner/user justifications in the joint. These relationships are in accordance with the visibility restrictions and the set of operations to whom the justificand is subjected and the justifier operates on. We can model a system as

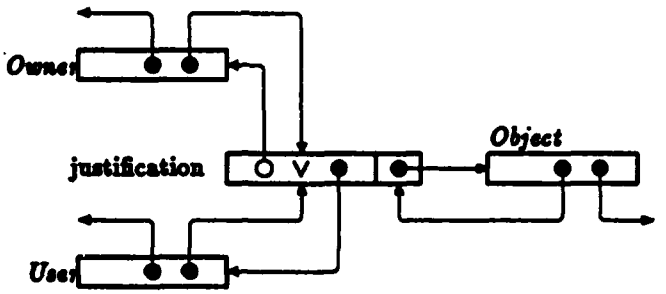


Figure 1: Object's Owner and User Justification

a graph whose nodes are objects and whose arcs are directed from justifier to justificand representing the owner/user relationship. An example is given in Figure 1.

In [3] three classes of operation types on objects are defined:

- Operations that change an object state.
- Operations that evaluate current state of an object.
- Operations that allow visiting parts of an object.

These operations can be carried out on object bodies as well as on object joints.

We have also discussed the distinction between on-line and off-line (infinite deadline) execution of objects. Scheduling executable objects and context initialization are each divided in our model into an on-line part and an off-line part. The context initializer consists of an off-line *allocator/binder* that manages the acceptance of jobs (requests to execute objects) and allocates the resources before loading and an on-line *loader* that activates schedulable objects that are invoked. The scheduling policy is managed by the *off-line scheduler* and the *on-line scheduler* carries out locally the policy, and dispatches loaded objects according to their time constraint. The verification of the schedulability of an executable object is a prime issue in this paper, and is expanded in section 3.

Executable objects can be divided into three type classes ([3]), depending on their relationships with other objects. Since our definition of an object is slightly different from that of software engineering, we modify this classification:

- *Actor*: An object which is subjected to no operation. It only operates on other objects.
- *Server*: An object which is only subjected to operations by other executable objects. It does not operate on other executable objects, but it may operate on non-executable objects.

- *Agent*: An object which operates on (one or more) executable objects on behalf of another executable object. In turn other executable objects may operate on it.

In addition there are the non-executable objects, which are only subjected to operations.

- *Passive*: An object which is only subjected to operations and does not operate on others.

The relationships between objects necessitate the grouping of objects of the same type into a meta-object, to whom the rest of the objects may refer to as a whole entity.

1.2 Time-Service in Hard Real-Time Systems

In our system model computations are described as graphs, where the vertices (nodes) are executable objects and the edges are relations between them. Being a real-time system, each of the active objects in the system is assumed to have an access to a "time-knowledge" resource, usually called a *clock*. Let $C_i(t)$ be the monotonic function that maps real-time (sometimes called global time or universal time) to clock-i time. The clocks in the system are *inaccurate* due to a nonzero drift-rate. This inaccuracy results in *incorrect* time readouts, in other words $C_i(t) \neq t$. Clock synchronization algorithms (for examples one may refer to [11,12,17,7,23,6]) result in having a local drift rate which is bounded. Therefore, each computation is assumed to be able to access a clock, and acquire the estimate of the current real-time $C_i(t) = t \pm \Delta_i(t)$, through the use of a *time server*.

Assuming that time servers use a linear clock interpretation, the service for a *get-time* request at computation node p is a construction of T_p, for $t \geq t_p^{(0)}$, by

$$T_p(t) = a_p(t)C_p(t) + b_p(t)$$

where $C_p(t)$ is the local clock which is periodically synchronized at least every J time units with the other clocks in the system. Let the synchronization local times be denoted by the sequence $t_p^{(i)}$. One possible definition of the coefficients $a_p(t)$, $b_p(t)$ is described in [13]:

- $t_p^{(0)} \leq t \leq t_p^{(1)}$: (initial coefficients)

$$a_p(t) = 1, b_p(t) = T_p^{(0)} - C_p^{(0)}(t_p^{(0)}).$$

- $t_p^{(i)} < t \leq t_p^{(i+1)}, i > 0$: (coefficients update)

$$a_p(t) = 1 + \frac{C_p^{(i)}(t_p^{(i)}) - T_p(t_p^{(i)})}{J}$$

$$b_p(t) = b_p(t_p^{(i-1)}) + \left(a_p(t_p^{(i-1)}) - a_p(t_p^{(i)}) \right) C_p^{(i)}(t_p^{(i)}).$$

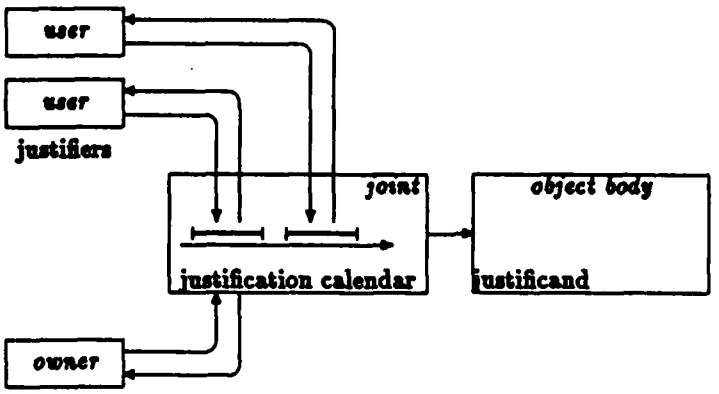


Figure 2: Object's Owner and User Temporal Justification

We define *event* as a detectable instantaneous atomic change in a system state. In a real-time system, the state of the system includes the set of time-servers $\{T_i(t)\}$. Let $T_i(t) = T_0$ be the set of system states in which the readout of time-server-i is T_0. Hence, we can define the following predicates on system states. The predicate $Taft(T_0)$ is *true* for $T_i(t) \geq T_0$ and *false* otherwise. The predicate $Tbef(T_d)$ is *true* for $T_i(t) \leq T_d$ and *false* otherwise.

Most of the time service algorithms (e.g. [13]) that have been published deal with two major issues. The first issue, synchronization or ordering of events in the system, may be associated with the past, in the sense that its requirements are due to events that have already occured. The major property required in this aspect is that any two clocks in the system must differ from each other in the lowest possible value. Hence, when reasoning about two events that are related to different local views of the global time, the proper resolution is obtained for event ordering. Thus, local histories that are based on the sequence of values assigned to local variables and on the sequence of local times at which these assignments took place[2] provide the means to analyze the past. The second issue, providing the knowledge of the global (universal) time, may be associated with the present. Here the service is to answer questions of the kind "what is the time *now*?". In that aspect, the major property required from each clock in the system is to be as correct as possible, or in other words as close to the global (universal) time. A third issue is the way in which a time service deals with projections onto the future. This issue is of extreme importance in hard real-time systems, in which some important decisions are taken in accordance with events that have not occured yet, but are known to occur in a known time interval in the future.

An example is given in Figure 2, describing a temporal justification scheme. The same server object is allocated to different users at different times, hence creating a user jus-

[2]Namely, relating event counters to an occurrence function while satisfying an accuracy axiom ([22,4]).

tification for this object at different time intervals. These intervals are in the future and according to them real-time scheduling decisions are to be taken. A very important issue arises in the above justification scheme. The time according to which the decisions are taken is a local and imprecise view of the global time. Distributed computations may have the same local view at different nodes at different "real" times. Therefore, future projection of time should avoid ambiguities and conflicts that originate in differences between local views.

This future projection issue is expanded in section 3.1 of this paper, but in order to have a better understanding of the needs, we first explain the way in which a guarantee to meet a deadline is examined in a real-time system.

1.3 Guarantees in Hard Real-Time Systems

When a request for a specific object invocation arrives at a hard real-time reactive operating system, the operating system has to allocate (in cases where it is feasible) all the resources required such that it is guaranteed that the object's time constraint is met. Informally speaking, a time constraint is a requirement to start executing a particular executable object, after a condition is satisfied, and complete the execution before its deadline. The execution time of the object is assumed to be given, and the constraint is extended to a periodic execution of the object. We have defined a time constraint formally in [15,16] as the quintuple

$$< Id, Taft(condition_1), c_{Id}, f_{Id}, Tbef(condition_2) >$$

where:

Id is the name of the executable object (process) in the proper context,

$Taft(condition_1)$ states after what event should execution begin. Simple *true* stands for "as soon as possible",

c_{Id} is the computation time of object Id,

f_{Id} is the frequency with which the computation should be carried out in case this is a periodic process. In case of a spontaneous (sporadic) process, this is the maximal frequency expected. $f_{Id} = 0$ stands for a single occurence of execution,

$Tbef(condition_2)$ states the deadline $d_{Id} = T_d - Now$ which should be met. $condition_2 \equiv T_{Id}(t) = T_d$ with $T_d = \infty$ stands for an "off line" computation.

The time interval defined by the events $Taft(condition_1)$ and $Tbef(condition_2)$ delimits the domain in which the executable object is allowed to execute.

A real-time reactive operating system uses the time constraint as the key to its decisions on execution initiation and resource scheduling ([18]). Before execution initiation, the allocation and context initialization are required to ensure schedulability of an accepted job. These tasks are carried out by:

- an off-line *allocator/binder* that manages the acceptance of jobs (requests to execute objects), the name binding and the allocation of the resources before loading, and
- an on-line *loader* that activates schedulable objects that are invoked.

In between the binding and the loading, a positive verification of the schedulability of the invoked object, according to its time constraint, is to be verified. All the required resources should be allocated and reserved for the object, assuring that it is going to meet its deadline. It is not only processors that are to be allocated for an object. An object may rely on other server objects in order to perform its functions. These other objects may or may not reside at the same site. Remote services necessitate the needs for agents and for communication, each of which has to be schedulable within the time constraints of the invoked object. One must take into account that time constraints are projected between different computation localities, such that each computation locality might have access to a different clock with a different accuracy and correctness. This projection is discussed in details in section 3.1.

The scheduling is also carried out in two parts.

- An *off-line scheduler* process manages the policy of a scheduling discipline. For example, when a resource is added to the system, in addition to recognizing the availability of that resource, a new discipline may be required.
- An *on-line scheduler* applies the current discipline.

Another major issue is the question of users sharing a server object, without violating the user objects' time constraints. Each server object is then considered as a resource, and maintains its own schedule. When a server is allocated to a new user, and the binder updates the justification links, the server's future schedule is to be checked to show schedulability within the new user's time constraints, without violating the services this server has already guaranteed to serve.

2 Notations

Having defined the system, we now introduce some tools with which temporal properties and relations can be expressed and analyzed.

2.1 Time Representation

The representation of time has been widely examined for purposes that vary from reasoning on planning in artificial intelligence to accurate scheduling in real-time operating systems. The characteristics we require from the representation to fit for reasoning about time, both for allocation (or planning) and for scheduling, are given below.

1. The representation of time should allow representing instantaneous events, expressing *dates* or *time points*.
2. It should allow representing events with durations, expressing named periods or *time intervals*.
3. It should suuport description of events whose duration may be *convex* (contiguous) or *non-convex* (containing gaps). Particularly, it should support representation of both *periodic* and *sporadic* non-convex event durations.
4. It should allow reasoning about a variety of temporal ordering relations.
5. It should support different granularity levels, allowing the resolution to vary according to the reasoning needs.
6. It should support relative as well as absolute quantifications.

2.2 Intervals Versus Points

The literature on time representation contains a long and conflicting debate, full of contradictory theories, between those who are in favour of a time point based representation, and those who are in favour of a time interval based representation. J. Allen (in [1]) has emphasized a major disadvantage of the time point representation of events. Since instantaneous events are not decomposable, time point events cannot be decomposed into subevents while maintaining an ordering. Furthermore, an event which seems to be instantaneous in one scope, may be decomposable in another scope.

On the other hand, using only intervals might allow a cumulative loss of time under some circumstances. For example, consider the operation of setting an interval timer by a "store" operation ([25]). The time from the clock interrupt to the actual store is lost. The above problem is usually solved by replacing the "store" by an "add". The overhead of clock update and the overhead of checking the task table after each interrupt constitute a lower bound on the clock granularity as well.

Yet, one should recall the fact that an interval can be represented by its endpoints. Therefore, allowing both points and intervals, along with allowing the granularity to vary, solve the above problems.

A real-time system needs both time interval and time point representations. The way in which a time constraint is defined in section 1.3 emphasizes it. Reasoning about a deadline requires absolute and relative time point representations. Considering and projecting the *Tbef* and *Taft* conditions, as well as the computation requirements, obviously requires time interval representations. The possibility of periodic time constraints extends the needs further to non-convex intervals ([9,10]), each of which might contain gaps.

2.3 Temporal Relations

2.3.1 Basic Definitions

The following definitions concern the entities with which we are going to reason about time, namely time points and time intervals.

Definition 1 *A* time point *is a real number that represents the occurrence time of an instantaneous event, and is an indivisible entity.* □

The order of two time points is defined if and only if the two time points are expressed with respect to the same reference frame. Let t_α and t_β be such time points.

Definition 2 *Two binary relations are defined for time points.*

- $t_\alpha < t_\beta$ *(t_α* before *t_β) and its inverse* $t_\alpha > t_\beta$ *(t_α* after *t_β).*
- $t_\alpha = t_\beta$. □

Definition 3 *A* convex time interval *is a contiguous period of time that specifies a rage of time points such that*

$$< t_\alpha, t_\beta > \equiv \{t : t_\alpha \leq t \leq t_\beta\}. \square$$

The above definition implies that a time point t_α can be associated with the time interval $< t_\alpha, t_\alpha >$.

Definition 4 *The* duration *of a convex time interval* $< t_\alpha, t_\beta >$ *is a measure of the interval such that*

$$\| < t_\alpha, t_\beta > \| = |t_\beta - t_\alpha|. \square$$

Definition 5 *A* non-convex time interval *is a set of subintervals expressed as an union of disjoint convex intervals.* □

The last definition means that unlike a convex interval, which is contiguous, a non-convex interval might contain gaps.

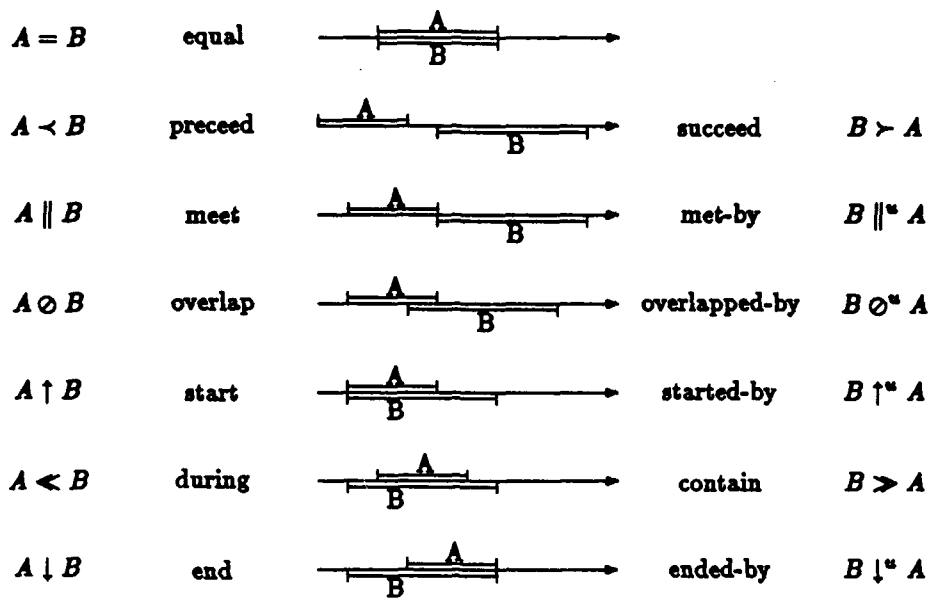


Figure 3: Convex Interval Binary Relations

2.3.2 Convex Interval Relations

A set of thirteen binary relations between convex intervals is proposed in [1]:

- *equal*,
- *precede* and its inverse *succeed*,
- *meet* and its inverse *met-by*,
- *overlap* and its inverse *overlapped-by*,
- *start* and its inverse *started-by*,
- *during* and its inverse *contain*,
- *end* and its inverse *ended-by*.

The relation *disjoint* ($\bowtie$) can therefore be expressed as

$$A \bowtie B \equiv (A \prec B) \vee (A \succ B),$$

and all the containment possibilities as

$$(A \uparrow B) \vee (A \ll B) \vee (A \downarrow B).$$

Equivalently, these relations can be defined based on the intervals' endpoints. A summary of these relations is given in Figure 3, using some notations borrowed from [9]. In [2], it is shown that one primitive, *meet*, is sufficient for constructing all the other relations.

2.3.3 Non-Convex Interval Relations

Our need to extend the relations of convex intervals to non-convex intervals, originates in the definition of a time constraint as given in section 1.3. Having a constraint whose frequency is lower than the reciprocal of the computation time, is directly modelled by a non-convex interval. In Definition 5 above, we defined a non-convex time interval by means of an operator applied to convex time intervals. In the following definitions let A and B be convex time intervals, such that

$$A =< t_\alpha^A, t_\beta^A >, B =< t_\alpha^B, t_\beta^B > .$$

We now define the application of some operators on these time intervals.

Definition 6 *The* interval intersection *of convex time intervals A and B is defined for $\neg(A \prec B) \wedge \neg(A \succ B)$ as*

$$A \sqcap B \equiv< \max(t_\alpha^A, t_\alpha^B), \min(t_\beta^A, t_\beta^B) >$$

and is defined ϕ (null) for $(A \prec B) \vee (A \succ B)$. □

Note that from the above definition, the intersection of two convex intervals that *meet* each other is a non-null convex interval of duration zero, or in other words a time point.

Definition 7 *The* cover *([14]) of convex time intervals A and B is a convex interval defined as*

$$A \uplus B \equiv< \min(t_\alpha^A, t_\alpha^B), \max(t_\beta^A, t_\beta^B) > .\square$$

The cover is a symmetric and commutative operation, a property that allows defining the operation on a set of more than two intervals

$$\biguplus_{i=1}^{n} \{c_i\} = c_1 \uplus c_2 \uplus c_3 \uplus \cdots \uplus c_n.$$

Definition 8 *The set of* maximal convex subintervals *of convex time intervals A and B is defined as $S(\{A\}, \{B\})$, such that*

- $A \sqcap B = \phi \Longrightarrow S = \{A, B\}$
- $A \sqcap B \neq \phi \Longrightarrow S = \{A \uplus B\}$.

The set of maximal convex subintervals of a non-convex time interval D is the set of maximal convex subintervals of all its convex members $\{d_i\}$. □

Definition 9 *The* set union *of non-convex time intervals C and D is a non-convex interval which consists of the set of members in $S(\{C\})$ and the set of members in $S(\{D\})$*

$$\{C\} \cup \{D\} \equiv S(\{C\}) \cup S(\{D\}).\square$$

Note that the cover of the set union

$$\biguplus(\{C\} \cup \{D\}) =< \min(t_\alpha^D, t_\alpha^C), \max(t_\beta^D, t_\beta^C) > .$$

The union is a symmetric and commutative operation, a property that allows defining the operation on a set of more than two intervals

$$\bigcup_{i=1}^{n}\{c_i\} = c_1 \cup c_2 \cup c_3 \cup \cdots \cup c_n.$$

Definition 10 *The* interval union *of non-convex time intervals C and D is a non-convex interval E, denoted*

$$E = C \sqcup D,$$

such that

$$E = S(\{C\} \cup \{D\}).\square$$

Note that the cover of the interval union equals the cover of the set union, and that the interval union is commutative too. Hence,

$$\bigsqcup_{i=1}^{n}\{c_i\} = c_1 \sqcup c_2 \sqcup c_3 \sqcup \cdots \sqcup c_n.$$

Applying generalization to the above convex interval relations, results in non-convex interval relations as described below. Let C and D be non-convex intervals, and let $\{c_i\}$ and $\{d_i\}$ be their corresponding sets of maximal convex subintervals. Let $\mathcal{R}$ and $\mathcal{Q}$ be convex relations defined in section 2.3.2.

- *mostly*: C mostly-$\mathcal{R}$ D if

$$\forall d_j \in D : \exists c_i \in C : c_i \mathcal{R} d_j.$$

- *always*: C always-$\mathcal{R}$ D if and only if C mostly-$\mathcal{R}$ D and D mostly-$\mathcal{R}^u$ C, where $\mathcal{R}^u$ is the converse relation to $\mathcal{R}$.
- *partially*: C partially-$\mathcal{R}$ D if and only if

$$X = \{x_i\} = \{c_i, d_i : c_i \mathcal{R} d_i\} \neq \phi,$$

$$\{C - X\} \sqcap \{D - X\} = \phi.$$

- *sometimes*: C sometimes-$\mathcal{R}$ D if and only if

$$X = \{x_i\} = \{c_i, d_i : c_i \mathcal{R} d_i\} \neq \phi.$$

- *disjunction*: C $\mathcal{R} \vee ... \vee \mathcal{Q}$ D if and only if

$$\forall c_i \in C, \forall d_j \in D : c_i \mathcal{R} d_j \vee ... \vee c_i \mathcal{Q} d_j \vee c_i \bowtie d_j.$$

- *totally*: C totally-$\mathcal{R}$ D if and only if

$$\forall d_j \in D : \forall c_i \in C : c_i \mathcal{R} d_j.$$

For example, the non-convex intervals disjoint ($\overline{\bowtie}$) relation, that can be expressed as "totally-$\bowtie$", meaning

$$\forall d_j \in D : \forall c_i \in C : c_i \bowtie d_j.$$

Further discussion on generalization of convex interval relations to non-convex interval relations and additional non-convex interval relations that are based on the leftmost and rightmost convex subintervals of non-convex intervals are provided in Appendix A. The leftmost convex subinterval of C is denoted $\triangleleft C$ and the rightmost one $\triangleright C$.

3 Schedulability

The above temporal relations serve as a basis for reasoning about scheduling. In this section we phrase conditions and propose mechanisms for a policy of accepting requests for invocations of objects (or resources), while guaranteeing that a deadline of an accepted request will be satisfied. An invocation of an object contains three phases of scheduling activities. In the first, the incoming time constraint is verified to be schedulable. In the second, the requesting user object chooses the computation localities for execution, from those where schedulability has been confirmed. In the third phase, an on-line selection and context switching is carried out. Some operating systems, e.g. CHAOS [20,21], merge

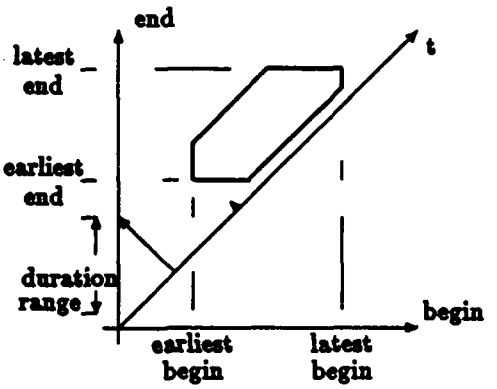


Figure 4: Time Constraint Laxity "Window"

the second and third phases into one. We find it very expensive to choose computation localities upon each context switching, and maintain three phases[3]. The first phase is carried out by an off-line *allocator*, the second by an on-line *loader*, and the third by an on-line *scheduler*.

3.1 Constraints Propagation

In section 1.3 we defined the system time constraints as

$$< Id, Taft(condition_1), c_{Id}, f_{Id}, Tbef(condition_2) > .$$

The conditions imposed on beginning, $Taft(condition_1)$, and on end, $Tbef(condition_2)$, create a time window whose duration must be at least c_{Id}. Generally this window is wider, such that upon execution there are more than one possibility that satisfy the constraint, creating a definable laxity of that constraint.

In [19], each time constraint is considered as a *set of possible occurrences* (SOPO), a domain in which the beginning, the end and the duration are constrained. A simple set of possible occurrences, when expressed graphically on begin-end axes as in Figure 4, creates a "window" of the constraint laxity. The diagonal axis, t, satisfies *begin* = *end*, or in other words represents the locus of dates. Obviously, the occurrences are confined to the area above the diagonal t axis, since *end* > *begin* for nonzero duration. Each point within the window satisfies the given time constraint: it starts after the earliest and before the latest begin time, it ends after the earliest and before the latest end time, and its duration varies from the minimal to the maximal duration time.

[3] See section 1.3.

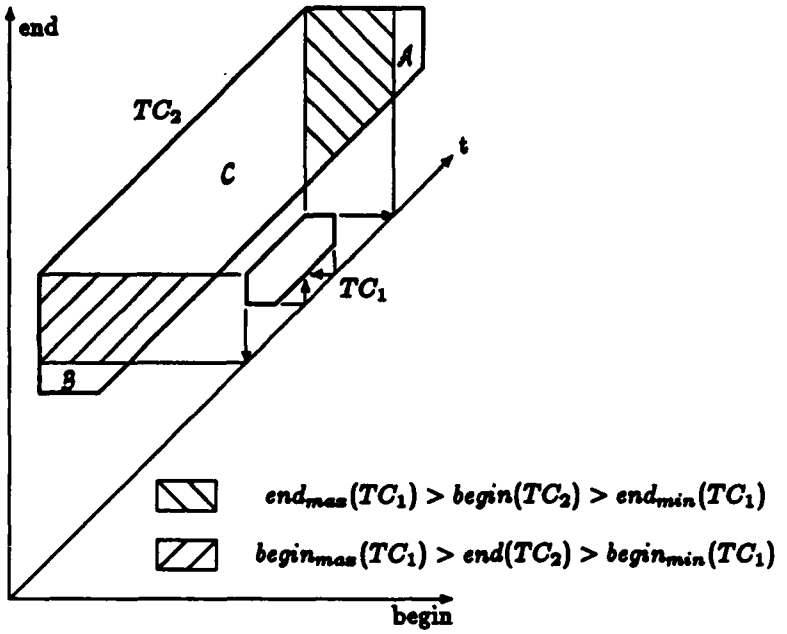


Figure 5: Time Constraint Propagation

When we have more than one constraint, the temporal relations between the constraints dictates the way in which constraints are propagated. Adhering to the notation of [19], we can examine the way in which a time constraint TC_1 is propagated onto another time constraint TC_2, as described in Figure 5. Recall that each point in the "window" of each time constraint stands for a possible occurrence with the appropriate $(begin, end)$ point coordinates. Projecting the latest possible $end(TC_1)$ onto the *begin* axis of TC_2, creates a window of occurrences, denoted $\mathcal{A}$, whose points stand for occurrences for which the relation $TC_1 \prec TC_2$ definitely holds. Projecting the earliest possible $begin(TC_1)$ onto the *end* axis of TC_2, creates the section $\mathcal{B}$, for which $TC_2 \prec TC_1$ definitely holds. Section $\mathcal{C}$ is characterized with

$$(begin(TC_2) < end_{min}(TC_1)) \wedge (end(TC_2) > end_{max}(TC_1)).$$

In other words, $TC_1 \sqcap TC_2$ is definitely non-null in section $\mathcal{C}$. The two streaked areas in Figure 5 show regions in which the temporal relation between the time constraints is not certain. In this regions the relation can be determined only upon an accurate knowledge of the actual $(begin, end)$ point.

The above method of constraint propagation has been demonstrated for a convex time constraint with relation to another convex time constraint. When a relation $\mathcal{R}_{i,j}$ is required

between TC_i and TC_j, then $TC_i \; \mathcal{R}_{i,j} \; TC_j$ is achieved if we intersect TC_j with the region allowed by TC_i and $\mathcal{R}_{i,j}$. Algorithms for propagation of convex constraints have been suggested for temporal reasoning purposes as well as for scheduling purposes, as in [1,24,19].

Extending the method for finite non-convex time intervals can be easily done for relations which are generated by application of generalization functors on convex relations. However, periodic time constraints, which consist of finite convex subintervals, yet are not finite non-convex intervals, are fairly difficult to be dealt with in this method. One possible simplification is the "local" solution, in which relations are derived and constraints are propagated only within a limited region, using the above methods. This solution solves many problems in which there is no need to reason about two constraints which are far apart from each other, as in our case.

Another propagation issue that is to be dealt with is the correct mapping of a constraint and its synchronized interpretation at different computation nodes. Being served by different clock servers means having different interpretations for an imposed time constraint. We assume that the time servers use a linear interpretation of the following type. The service for a *get-time* request is

$$T_p(t) = a_p(t)C_p(t) + b_p(t) \; , \; t \geq t_p^{(0)}$$

as defined in section 1.2. The bounds on the correct knowledge of $a_p(t)$ and $b_p(t)$ set the scale in which a projection of a time constraint is propagated. The projection scale must ensure that no violation of the constraint will occur due to the projection.

Let $\Delta a_p \equiv \max|1 - a_p(end_{max}(TC_i))|$ and let $\Delta b_p \equiv \max|b_p(end_{max}(TC_i))|$. Let $\delta a_p \equiv \max|1 - a_p(begin_{min}(TC_i))|$ and let $\delta b_p \equiv \max|b_p(begin_{min}(TC_i))|$. Accordingly, at a computation node p an imposed time constraint TC_i maps to the local bounds:

$$begin'_{min}(TC_i) = begin_{min}(TC_i) + \delta a_p begin_{min}(TC_i) + \delta b_p,$$

$$begin'_{max}(TC_i) = begin_{max}(TC_i) - \delta a_p begin_{max}(TC_i) - \delta b_p,$$

$$end'_{min}(TC_i) = end_{min}(TC_i) + \Delta a_p end_{min}(TC_i) + \Delta b_p,$$

$$end'_{max}(TC_i) = end_{max}(TC_i) - \Delta a_p end_{max}(TC_i) - \Delta b_p.$$

The laxity is reduced, but no violation of the constraint can happen due to clock inaccuracies.

In the rest of this paper we implicitly assume that the above mapping is applied to every incoming time constraint before accepting it.

3.2 Allocation and Scheduling Orientation

For allocation verification purposes and for scheduling purposes, we may decrease the degree of generality, that has been defined in the previous section, by considering only the maximal durations of the constraints. The reason of decreasing the generality, which has been increased primarily for temporal reasoning purposes, is that for verification of the schedulability of a new object one needs to assume the worst case in order to be able to guarantee future execution. Considering the possible allocations shows that the worst case means maximal time consumption by the already accepted objects.

Let the j'th occurrence of time constraint i be denoted as $TC_i^{(j)}$, represented as the j'th maximal convex subinterval of non-convex interval TC_i. Let $P_i^{(j)}$ be a convex interval, for which

$$(P_i^{(j)} \uparrow TC_i^{(j)}) \vee (P_i^{(j)} \ll TC_i^{(j)}) \vee (P_i^{(j)} \downarrow TC_i^{(j)}).$$

The definition of a time constraint in section 1.3 can therefore be converted as follows.

- $Taft(condition_1)$[4] $\longrightarrow$ $begin_{min}(TC_i^{(j)})$.
- $Tbef(condition_2)$[5] $\longrightarrow$ $end_{max}(TC_i^{(j)})$.
- f_i periodicity $\longrightarrow$ $\forall j > 1 : end_{max}(TC_i^{(j)}) - end_{max}(TC_i^{(j-1)}) = \frac{1}{f_i}$.
- c_i computation time $\longrightarrow$ $\forall j \geq 1 : \|P_i^{(j)}\| = c_i$.

The fact that the duration of $P_i^{(j)}$ might be less than that of $TC_i^{(j)}$ requires a more precise definition regarding the degree of freedom we have in moving $P_i^{(j)}$ within the $TC_i^{(j)}$ window.

Definition 11 *The* laxity *of a (computation) convex interval $P_i^{(j)}$ that is constrained within a (window) convex interval $TC_i^{(j)}$ is defined by the pair* (backward_slack,forward_slack). *If no other constraint is imposed, then*

$$backward\_slack = x_- = t_\alpha^P - t_\alpha^{TC},$$

$$forward\_slack = x_+ = t_\beta^{TC} - t_\beta^P.$$

Imposition of additional constraints that affect the $TC_i^{(j)}$ window should be considered when the laxity is derived. □

[4]Begin filter.

[5]Deadline.

Let TC_{in} be a time constraint whose schedulability is to be tested. Let all the other already accepted time constraints, $TC_i^{(j)}$, be schedulable. We first consider the case of a single-occurrence time constraint with a contiguous computation requirement (hence convex), and then we analyze the bounds that guarantee schedulability of a non-convex time constraint.

3.2.1 Convex Time Constraints

A single-occurrence time constraint has a known window where it might occur:

$$TC_{in} =< begin_{min}(TC_{in}), end_{max}(TC_{in}) > .$$

In this window one must verify that after satisfying the already accepted time constraints one can schedule a convex subinterval of duration $\|P_{in}\|$ which is contained within TC_{in}. The already accepted time constraints are assumed schedulable already. From these already accepted time constraints construct a set of maximal convex subintervals, which are checked according to our *verification interval*

$$V =< t_\alpha^V, t_\beta^V > .$$

Initially, $V = TC_{in} =< begin_{min}(TC_{in}), end_{max}(TC_{in}) >$. Each subinterval that intersects with V, denoted $TC_i^{(j)}$, has one of four possible relations with it.

1. $TC_i^{(j)} \oslash V$, or
2. $(TC_i^{(j)} \uparrow V) \vee (TC_i^{(j)} \ll V) \vee (TC_i^{(j)} \downarrow V)$, or
3. $TC_i^{(j)} \oslash^u V$, or
4. $(TC_i^{(j)} \uparrow^u V) \vee (TC_i^{(j)} \gg V) \vee (TC_i^{(j)} \downarrow^u V)$.

Let the set of all the above interfering maximal convex subintervals be the *interfering set*, denoted I. Let I_1 be the subset of those that overlap V, I_3 the subset of those that are overlapped by V, I_2 the subset of those that are within V, and I_4 the subset of those containing V. If I_4 is empty, then we verify directly according to the boundaries of TC_{in}. Otherwise, we extend the verification boundaries to those of a member of I_4 which is not contained within any other member of I_4. This extension requires appropriate extensions of I_1, I_2 and I_3. Each of the members of I_4 is moved to the appropriate subset as well. The constraint which sets the boundaries of V is of course in I_2. For worst case analysis we consider an artificial case in which

$$TC_i^{(j)} \in I_1 : \; P_i^{(j)} \downarrow TC_i^{(j)}$$

setting $x_+^{TC_i^{(j)}} \leftarrow 0$, and

$$TC_i^{(j)} \in I_3 : P_i^{(j)} \uparrow TC_i^{(j)}$$

setting $x_-^{TC_i^{(j)}} \leftarrow 0$. Doing so, we encounter the largest computation requirement possible within the verification window.

We can now create two new subsets I_1^o and I_3^o, for which only the relevant portions of overlapping time constraints are accounted for.

$$I_1^o \equiv \{ TC_{i_o}^{(j_o)} \} = \{ < \max(begin_{min}(TC_i^{(j)}), t_a^V), end_{max}(TC_i^{(j)}) >, TC_i^{(j)} \in I_1 \}$$

$$I_3^o \equiv \{ TC_{i_o}^{(j_o)} \} = \{ < begin_{min}(TC_i^{(j)}), \min(t_\beta^V, end_{max}(TC_i^{(j)})) >, TC_i^{(j)} \in I_3 \}$$

For each of the subintervals in I_1^o and I_3^o the computation requirement $P_i^{(j)}$ is appropriately set to

$$\|P_{i_o}^{(j_o)}\| = \min(\|P_i^{(j)}\|, \|TC_{i_o}^{(j_o)}\|).$$

Hence, we can formalize the condition for schedulability for both preemptive and non-preemptive schedulers.

Condition 1 *TC_{in} is* non-preemptively schedulable *if*

$$\forall i \forall j : TC_i^{(j)} \in I^o : \exists x_-^{TC_i^{(j)}} \geq 0. \exists x_+^{TC_i^{(j)}} \geq 0. \exists x_-^{TC_{in}} \geq 0. \exists x_+^{TC_{in}} \geq 0. :$$

$$P_{in} \bowtie P_i^{(j)}$$

where I^o is the set union of I_1^o, I_2 and I_3^o. In other words,

$$\{P_{in}\} \overline{\bowtie} P^o$$

where P^o is the set $\{P_i^{(j)} : TC_i^{(j)} \in I^o\}$. □

An example of such a verification, which obeys Condition 1 is given in Figure 6. The verification interval (V) is TC_{in}, because I_4 is empty. I_1 contains one convex subinterval of TC_1, I_3 contains one convex subinterval of TC_3 and I_2 contains two convex subintervals of TC_2. As can be seen in this example, P_{in} demonstrates non-preemptive schedulability, since it is disjoint from the disjoint occurrences of the relevant $P_i^{(j)}$'s (i=1,2,3).

The totally-disjoint ($\overline{\bowtie}$) requirement can hold either if the computation interval P_{in} is disjoint in its original placement, or if TC_{in} has enough laxity to allow a new placement of P_{in} to give a disjoint relation. The time constraints of I^o that have either overlapping or containment relations with TC_{in}, affect its laxity. However, their own laxities are affected

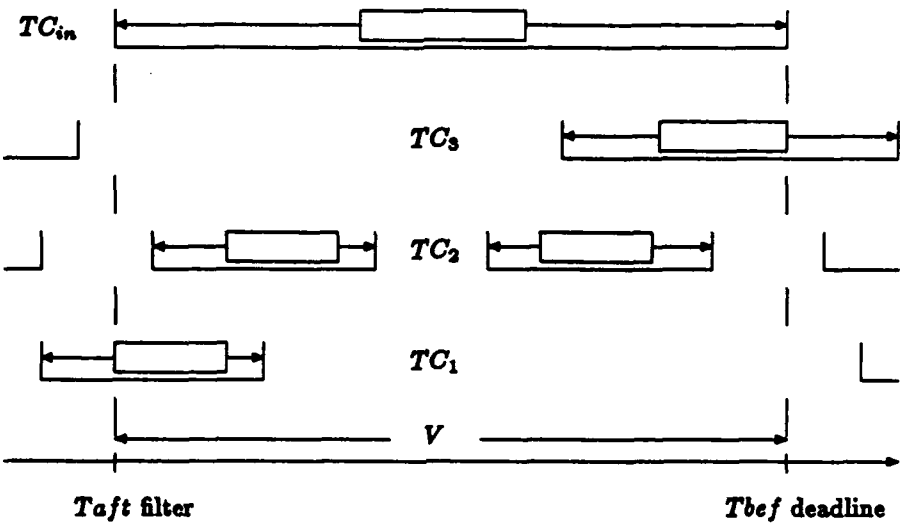


Figure 6: Verification of Schedulability of TC_{in}

by TC_{in} as well. Figure 7 demonstrates a case of two overlapping constraints, $TC_A \oslash TC_B$, whose laxities allow the disjoint relations. The effects on the laxities are as follows.

$$x_-'^B = \min(\, x_-^B, \min(\, t_\alpha^{P_B} - t_\alpha^{TC_B}, t_\alpha^{P_B} - (t_\beta^{P_A} - x_-^A)\,)\,),$$

since P_B is constrained by P_A from the left. If $x_-'^B$ is negative, and P_B can be moved to the right (on x_+^B's account), the disjoint relation still holds. This requires

$$x_-'^B + x_+^B > 0.$$

However, P_A is constrained by P_B from the right. Hence,

$$x_+'^A = \min(\, x_+^A, \min(\, t_\beta^{TC_A} - t_\beta^{P_A}, (t_\alpha^{P_B} + x_+^B) - t_\beta^{P_A}\,)\,).$$

Here again, a negative $x_+'^A$ still produces the disjoint relation, by moving P_A to the left, if

$$x_+'^A + x_-^A > 0.$$

In a preemptive schedule the condition can be less demanding, in the sense that the interval allocated for TC_{in} does not have to satisfy convexity, and therefore only duration constraints must be imposed. In order to analyze the requirements, we examine two time constraints TC_A and TC_B, that contain computation requirements P_A and P_B respectively.

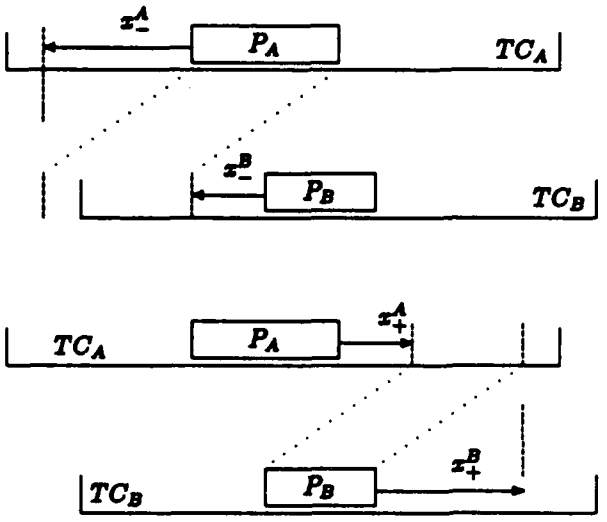


Figure 7: Laxity Interaction of Overlapping Time Constraints

If the context swiching can be regarded as negligible, then one can allocate a *resource portion* (γ_A,γ_B) per time unit during the time intervals TC_A and TC_B respectively, such that

$$\gamma_A = \frac{\|P_A\|}{\|TC_A\|} \leq 1$$

$$\gamma_B = \frac{\|P_B\|}{\|TC_B\|} \leq 1$$

and still satisfy the requirements. In other words, satisfication is due to

$$\int_{TC_A} \gamma_A \, dt = P_A$$

and respectively for P_B.

Yet, when two or more constraints are not disjoint, one must satisfy an additional constraint. Resource portions cannot be allocated in a manner that excedes a unity for a single resource. In other words, in all possible instances of resource requirement intersection the following may be asked to hold.

$$\gamma_A + \gamma_B + \cdots \leq 1$$

However, this requirement is too strong for the following reasons. First, the assumption that the resource portion is a constant is not necessary. Second, even for a non-constant

$\gamma(t)$ the convexity assumption is not necessary, since we allow preemption. Therefore, a resonable condition can be derived by checking that for intersecting requirement intervals a single resource (or the inventory in the multiple resource bank case) has the "capacity" to respond. For intersecting TC_A, TC_B, and others, it can be formulated as

$$\int_{TC_A} \gamma_A(t)\,dt + \int_{TC_B} \gamma_B(t)\,dt + \cdots = \|P_B\| + \|P_B\| + \cdots \leq \|TC_A \sqcup TC_B \sqcup \cdots\|.$$

Recalling the definition of the set of maximal convex subintervals[6], the condition we impose for accepting a time constraint TC_{in} is given below.

Condition 2 *Let V be the verification interval, derived from the duration of a time constraint TC_z, for which $TC_z = TC_{in}$ or $TC_z \gg TC_{in}$. Let I_V be*

$$I_V = I^o \cup \{TC_{in}\} - \{TC_z\}$$

where I^o is the set union of I_1^o, I_2 and I_3^o.
TC_{in} is preemptively schedulable *if*

$$\forall s_v \in S(I_V) : \sum_{\forall i: TC_i \in I_V} \|s_v \sqcap P_i\| \leq \|s_v\|$$

$$\bigwedge$$

$$\sum_{\forall s_v \in S(I_V)} \|s_v\| \leq \|V\| - \|P_z\|$$

where s_v are the convex subintervals of $S(I_V)$. □

An algorithm for verification of the schedulability of a convex constraint is proposed in section 3.3.1, based on the above conditions. The properties of that algorithm are discussed in section 4.

3.2.2 Non-convex Time Constraints

In real systems, time constraints might have some arbitrary precedence relations between them. Grouping them into one time constraint, while forcing their execution to be carried out in a particular computation node ([5]), imposes a non-convex resource requirement on that node. This non-convex resource requirement is the set of maximal convex subintervals of the group individual resource requirements. The time window in which this group is allowed to execute is the cover of the individual windows of the group. The result of

[6] See Definitions 8 and 10.

the grouping is therefore a non-convex time constraint. That fact is a good motivation to extend the convex constraint schedulability conditions to the finite non-convex case. Furthermore, a resource requirement might be noncontinuous to begin with. A better utilization of the resource is expected if it can be released in the gaps between the subintervals in which it is needed, without conflicting these subintervals.

Formally, a non-convex time constraint differs from a convex one in the nature of its noncontiguous computation interval. For an occurrence window TC_i corresponds a non-convex computation interval $P_i = \bigcup_j P_i^{(j)}$, such that

$$(\biguplus_j P_i^{(j)} \uparrow TC_i) \vee (\biguplus_j P_i^{(j)} \ll TC_i) \vee (\biguplus_j P_i^{(j)} \downarrow TC_i).$$

The definition of laxity is slightly extended to include the non-convex nature of the computation requirement.

Definition 12 *The* laxity *of a (computation) non-convex interval P_i that is constrained within a (window) convex interval TC_i is defined by the pair $(x_-^{TC_i}, x_+^{TC_i})$, such that*

$$x_-^{TC_i} \leq t_a^{P_i^L} - t_a^{TC_i} = t_a^{\uplus P_i} - t_a^{TC_i},$$

$$x_+^{TC_i} \leq t_\beta^{TC_i} - t_\beta^{P_i^R} = t_\beta^{TC_i} - t_\beta^{\uplus P_i},$$

where $P_i^L = \lhd P_i$ and $P_i^R = \rhd P_i$. $\square$.

If no other constraint is imposed then the equality holds . Additional constraints that intersect with TC_i may reduce the laxity and then the inequality holds. In case such a reduction occurs, it is done in the same way it is defined for a convex time constraint (and illustrated in Figure 7), since the laxity is defined above with respect to $\uplus P_i$, and $\uplus P_i$ is convex.

The schedulability is verified within the time interval where the computation is allowed to occur. In this window one must verify that after satisfying the already accepted time constraints one can satisfy the constraint that is to be verified.

The construction of the *interfering set* is similar to the process defined for convex time constraints, since the occurrence window is convex here too. Hence, the subsets I_1 to I_4 are the same in the case of a non-convex time constraint. The worst case assumptions are taken here too, but here they are based on non-convex relations between convex occurrence windows TC_i and their corresponding computation non-convex intervals P_i.

$$\forall TC_i \in I_1 : P_i \hat{\updownarrow} \{TC_i\}$$

setting $x_+^{TC_i} \leftarrow 0$, and

$$\forall TC_i \in I_3 : P_i \hat{\uparrow} \{TC_i\}$$

setting $x_-^{TC_i} \leftarrow 0$. Worst case is expressed here by having the largest computation requirement possible within the verification window, as in the convex time constraint case. Still, I_1 and I_3 include portions which are outside the verification window. Therefore, subsets which contain only the portions of computation that are in the required verification window are created.

$$I_1^o \equiv \{ TC_{i_o} \} = \{ < \max(begin_{min}(TC_i), t_\alpha^V), end_{max}(TC_i) >, TC_i \in I_1 \}$$

$$I_3^o \equiv \{ TC_{i_o} \} = \{ < begin_{min}(TC_i), \min(t_\beta^V, end_{max}(TC_i)) >, TC_i \in I_3 \}$$

For each of the subintervals in I_1^o and I_3^o the non-convex computation requirement P_i is appropriately set to

$$P_{i_o} = \{ \bigcup_j P_{i_o}^{(j)} \mid P_{i_o}^{(j)} = P_i^{(j)} \sqcap TC_{i_o},\ P_i^{(j)} \in P_i \}.$$

As in the convex time constraint case, a basic assumption taken here is that the already accepted time constraints are already schedulable. Based on this assumption and on the above relations of the interfering set and the verification window, we now formulate the conditions for schedulability of a non-convex time constraint.

First, for the non-preemptive schedule, we require a totally-$\bowtie$ relation between all the computations, as well as avoiding conflicts between windows of occurrence and their corresponding computation non-convex intervals.

Condition 3 *A time constraint with an occurrence window TC_{in} and a non-convex computation requirement P_{in} is* non-preemptively schedulable *if*

$$\forall i \forall j : TC_i^{(j)} \in I^o : \exists x_-^{TC_i^{(j)}} \geq 0. \exists x_+^{TC_i^{(j)}} \geq 0. \exists x_-^{TC_{in}} \geq 0. \exists x_+^{TC_{in}} \geq 0. :$$

$$\forall P_{i_o}^{(j)} \in P^o : P_{in} \hat{\bowtie} P_{i_o}^{(j)}$$

where I^o is the set union of I_1^o, I_2 and I_3^o and P^o is the set $\{P_{i_o}^{(j)} \mid TC_i^{(j)} \in I^o\}$. □

In the prteemptive schedule, we now take into account the non-convex nature of the computation intervals.

Condition 4 *Let the new constraint be a time constraint with an occurrence window TC_{in} and a non-convex computation requirement P_{in}. Let V be the verification interval, derived*

```
typedef struct convex_time_interval {
        float start ;
        float fin ;
        float reference_scale ;
        float reference_bias ;
} ;
typedef struct time_constraint {
        struct convex_time_interval occurrence_window ;
        float backward_slack, forward_slack ;
        struct convex_time_interval computation_window ;
        float frequency ;
        int state ;
        struct time_constraint *successor ;
        struct time_constraint *predecessor ;
} ;
```

Figure 8: Convex Time Constraint Expressed in C

from the duration of a time constraint TC_x, for which $TC_x = TC_{in}$ or $TC_x \gg TC_{in}$. Let I_V be

$$I_V = I^o \cup \{TC_{in}\} - \{TC_x\}$$

where I^o is the set union of I_1^o, I_2 and I_3^o.
TC_{in} is preemptively schedulable *if*

$$\forall s_v \in S(I_V) : \sum_{\forall i: TC_i \in I_V} \sum_{\forall k: P_i^{(k)} \in P_i} \|s_v \sqcap P_i^{(k)}\| \leq \|s_v\|$$

$$\bigwedge$$

$$\sum_{\forall s_v \in S(I_V)} \|s_v\| \leq \|V\| - \sum_{\forall k: P_x^{(k)} \in P_x} \|P_x^{(k)}\|$$

where s_v are the convex subintervals of $S(I_V)$. □

3.3 Scheduling Feasibility Verification Algorithms

Having defined the conditions for schedulability for incoming time constraints, one needs to define mechanisms that provide the ability of verifying these properties. The mechanisms

consist of data structures and algorithms. In section 1.3 we have proposed a set of modules that perform the allocation of a resource or an object. Upon a request for allocation, the local *allocator* verifies the schedulability of the request in the local *calendar* and passes the request to other resources or objects if they are needed. The knowledge of the resource requirements is taken from the *joint* of the invoked object. If the requests it has passed are confirmed along with a verification of local schedulability, it reserves the local resource. The reservation is for an a priori decided time, and the requesting node must acknowledge the reservation within this time by invoking the *loader*.

In the above concept a *calendar* is a set of accepted and reserved time constraints. In section 3.2 we have shown an equivalent representation which is a time interval based representation. An example of an implementation of such a data structure is given in Figure 8.

Throughout the rest of this section we propose two algorithms based on that data structure and on the conditions of schedulability proposed above. The first algorithm, in section 3.3.1, is a boolean function that installs a convex time constraint into a calendar if its schedulability is verified according to conditions 1 and 2. The second algorithm, in section 3.3.2, is an extension of the first one to install a nonconvex time constraint into a calendar if its schedulability is verified according to conditions 3 and 4. Although not explicitly stated in the algorithm, the access to the global variable that represents the current calendar must be protected for mutual execlusion, to avoid concurrent accesses and false deduction. The PUSH_TC boolean function we propose is one of the service access points (SAPs) of the scheduler. Other SAPs might access the same calendar concurrently. In the system concept we proposed above, this mutual exclusion mechanism is a part of the *object* access mechanism of the *scheduler* that is implemented in its *joint*.

3.3.1 Convex Constraint Allocation Algorithm

```
type class = { preemptive,non_preemptive } ;
type convex_time_interval= construct
                    {       begin: time_point ;
                            end: time_point ;
                            scale, bias: real } ;
type time_constraint = construct
                    {       tc: convex_time_interval ;
                            back_slack, for_slack : real ;
                            P : convex_time_interval ;
                            freq : real ;
                            state : integer } ;
global var Already_Accepted: set of time_constraints;

boolean function scheduler.PUSH_TC (TC_in:time_constraint, schedule_type: class) ;

local var I^o, I_1, I_2, I_3, I_4, I_1^o, I_3^o, I_V: set of time_constraint ;
      TC_i^(j), TC_x, constraint: time_constraint ;
      S_V: set of convex_time_interval ;
      v, s_v: convex_time_interval ;
      i, j, i_o, j_o, k, n: indices ;

/* Init */
TC_x ← TC_in ;
v ← TC_in.tc ;
I^o ← φ ;

/* Prepare verification interval and classify interference set
      of already accepted time constraints. */
repeat
      I_1, I_2, I_3, I_4, I_1^o, I_3^o ← φ ;
      i ← 0 ;
      ∀constraint ∈ Already_Accepted Do
            if constraint.freq ≠ 0. then Do
                  j ← ⌈(v.begin − constraint.tc.end)/constraint.freq⌉ ;
                  TC_i^(j).tc.begin ← constraint.tc.begin + j/constraint.freq ;
                  TC_i^(j).tc.end ← constraint.tc.end + j/constraint.freq ;
                  oD ;
            else Do
                  j ← 0 ;
                  TC_i^(j) ← constraint ;
                  oD ;
```

/* Classification switch */

while $TC_i^{(j)}.tc.begin < v.end$ Do
$TC_i^{(j)}.tc \oslash v \Longrightarrow I_1 \leftarrow I_1 \cup \{TC_i^{(j)}\}$;
$(TC_i^{(j)}.tc \uparrow v) \vee (TC_i^{(j)}.tc \ll v) \vee (TC_i^{(j)}.tc \downarrow v) \Longrightarrow I_2 \leftarrow I_2 \cup \{TC_i^{(j)}\}$;
$TC_i^{(j)}.tc \oslash^u v \Longrightarrow I_3 \leftarrow I_3 \cup \{TC_i^{(j)}\}$;
$(TC_i^{(j)}.tc \uparrow^u v) \vee (TC_i^{(j)}.tc \gg v) \vee (TC_i^{(j)}.tc \downarrow^u v) \Longrightarrow I_4 \leftarrow I_4 \cup \{TC_i^{(j)}\}$;

/* Next instance of periodic constraints */
if $constraint.freq \neq 0.$ then Do
$j \leftarrow j + 1$;
$TC_i^{(j)}.tc.begin \leftarrow constraint.tc.begin + j/constraint.freq$;
$TC_i^{(j)}.tc.end \leftarrow constraint.tc.end + j/constraint.freq$;
oD
else
break ;
oD ;
$i \leftarrow i + 1$;
oD ;
if $I_4 \neq \phi$ then Do
$TC_x \leftarrow max\_convex\_constraint(I_4)$;
$v \leftarrow max\_convex\_interval(I_4)$;
oD ;
until $I_4 = \phi$;

/* Update the overlapping already-accepted constraints (I_1, I_3)
to contain only relevant portion within the verification interval. */

$\forall i \forall j : TC_i^{(j)} \in I_1$: Do
$TC_{i_o}^{(j_o)}.tc.begin \leftarrow \max(v.begin, TC_i^{(j)}.tc.begin)$;
$TC_{i_o}^{(j_o)}.tc.end \leftarrow TC_i^{(j)}.tc.end$;
$I_1^o \leftarrow I_1^o \cup \{TC_{i_o}^{(j_o)}\}$;
oD ;
$\forall i \forall j : TC_i^{(j)} \in I_3$: Do
$TC_{i_o}^{(j_o)}.tc.end \leftarrow \min(TC_i^{(j)}.tc.end, v.end)$;
$TC_{i_o}^{(j_o)}.tc.begin \leftarrow TC_i^{(j)}.tc.begin$;
$I_3^o \leftarrow I_3^o \cup \{TC_{i_o}^{(j_o)}\}$;
oD ;
$I^o \leftarrow \{TC_i^{(j)} : (TC_i^{(j)} \in I_1^o) \vee (TC_i^{(j)} \in I_2) \vee (TC_i^{(j)} \in I_3^o)\}$;

/* Check schedulability condition and accept if possible. */

if *schedule_type*=non_preemptive **then Do**
  if (*result* $\leftarrow \forall i \forall j : TC_i^{(j)} \in I^o :$
    $\exists TC_i^{(j)}.back\_slack \geq 0. \wedge \exists TC_i^{(j)}.for\_slack \geq 0. \wedge \exists TC_{in}.back\_slack \geq 0. \wedge \exists TC_{in}.for\_slack \geq 0. :$
    $TC_{in}.P \bowtie TC_i^{(j)}.P$) **then**
      $Already\_Accepted \leftarrow Already\_Accepted \cup \{TC_{in}\}$;
  return(*result*) ;
oD ;
if *schedule_type*=preemptive **then Do**
  $S_V \leftarrow S(I_V \leftarrow I^o \cup \{TC_{in}\} - \{TC_n\})$;
  if (*result* $\leftarrow \forall s_v \in S_V :$
    $(\forall i : TC_i \in I_V : \sum_i \|P_i \sqcap s_v\| \leq \|s_v\|)$
    $\wedge (\sum \|s_v\| \leq \|v\| - \|TC_n.P\|)$) **then**
      $Already\_Accepted \leftarrow Already\_Accepted \cup \{TC_{in}\}$;
  return(*result*) ;
oD ;
□

3.3.2 Non-Convex Constraint Allocation Algorithm

```
type class = { preemptive,non_preemptive } ;
type convex_time_interval= construct
                { begin: time_point ;
                  end: time_point ;
                  scale, bias: real } ;
type non_convex_time_interval = set of convex_time_interval ;
type time_constraint = construct
                { tc: convex_time_interval ;
                  back_slack, for_slack : real ;
                  P : non_convex_time_interval ;
                  freq : real ;
                  state : integer } ;
global var Already_Accepted: set of time_constraints;
boolean function scheduler.PUSH_TC (TC_in:time_constraint, schedule_type: class) ;
local var I°, I_1, I_2, I_3, I_4, I°_1, I°_3, I_V: set of time_constraint ;
      P°: set of non_convex_time_interval ;
      TC_i^(j), TC_s, constraint: time_constraint ;
      S_V: set of convex_time_interval ;
      s_v, v: convex_time_interval ;
      i, j, i_o, j_o, k, n: indices ;

/* Init */
TC_s ← TC_in ;
v ← TC_in.tc ;
I° ← φ ;

/* Prepare verification interval and classify interference set
      of already accepted time constraints. */
repeat
      I_1, I_2, I_3, I_4, I°_1, I°_3 ← φ ;
      i ← 0 ;
      ∀constraint ∈ Already_Accepted Do
            if constraint.freq ≠ 0. then Do
                  j ← ⌈(v.begin − constraint.tc.end)/constraint.freq⌉ ;
                  TC_i^(j).tc.begin ← constraint.tc.begin + j/constraint.freq ;
                  TC_i^(j).tc.end ← constraint.tc.end + j/constraint.freq ;
                  oD ;
            else Do
                  j ← 0 ;
                  TC_i^(j) ← constraint ;
                  oD ;
```

/* Classification switch */
while $TC_i^{(j)}.tc.begin < v.end$ Do
$TC_i^{(j)}.tc \oslash v \Longrightarrow I_1 \leftarrow I_1 \cup \{TC_i^{(j)}\}$;
$(TC_i^{(j)}.tc \uparrow v) \vee (TC_i^{(j)}.tc \ll v) \vee (TC_i^{(j)}.tc \downarrow v) \Longrightarrow I_2 \leftarrow I_2 \cup \{TC_i^{(j)}\}$;
$TC_i^{(j)}.tc \oslash^u v \Longrightarrow I_3 \leftarrow I_3 \cup \{TC_i^{(j)}\}$;
$(TC_i^{(j)}.tc \uparrow^u v) \vee (TC_i^{(j)}.tc \gg v) \vee (TC_i^{(j)}.tc \downarrow^u v) \Longrightarrow I_4 \leftarrow I_4 \cup \{TC_i^{(j)}\}$;
/* Next instance of periodic constraints */
if $constraint.freq \neq 0.$ then Do
$j \leftarrow j + 1$;
$TC_i^{(j)}.tc.begin \leftarrow constraint.tc.begin + j/constraint.freq$;
$TC_i^{(j)}.tc.end \leftarrow constraint.tc.end + j/constraint.freq$;
oD
else
break ;
oD ;
$i \leftarrow i + 1$;
oD ;
if $I_4 \neq \phi$ then Do
$TC_s \leftarrow max\_convex\_constraint(I_4)$;
$v \leftarrow max\_convex\_interval(I_4)$;
oD ;
until $I_4 = \phi$;
/* Update the overlapping already-accepted constraints (I_1, I_3)
to contain only relevant portion within the verification interval. */
$\forall i \forall j : TC_i^{(j)} \in I_1$: Do
$TC_{i_o}^{(j_o)}.tc.begin \leftarrow \max(v.begin, TC_i^{(j)}.tc.begin)$;
$TC_{i_o}^{(j_o)}.tc.end \leftarrow TC_i^{(j)}.tc.end$;
$TC_{i_o}^{(j_o)}.for\_slack \leftarrow 0.$;
$TC_{i_o}^{(j_o)}.P \leftarrow TC_i^{(j)}.P \cap TC_{i_o}^{(j_o)}.tc$;
$I_1^o \leftarrow I_1^o \cup \{TC_{i_o}^{(j_o)}\}$;
oD ;
$\forall i \forall j : TC_i^{(j)} \in I_3$: Do
$TC_{i_o}^{(j_o)}.tc.end \leftarrow \min(TC_i^{(j)}.tc.end, v.end)$;
$TC_{i_o}^{(j_o)}.tc.begin \leftarrow TC_i^{(j)}.tc.begin$;
$TC_{i_o}^{(j_o)}.back\_slack \leftarrow 0.$;
$TC_{i_o}^{(j_o)}.P \leftarrow TC_i^{(j)}.P \cap TC_{i_o}^{(j_o)}.tc$;
$I_3^o \leftarrow I_3^o \cup \{TC_{i_o}^{(j_o)}\}$;
oD ;
$I^o \leftarrow \{TC_i^{(j)} : (TC_i^{(j)} \in I_1^o) \vee (TC_i^{(j)} \in I_2) \vee (TC_i^{(j)} \in I_3^o)\}$;
$\forall i \forall j : TC_i^{(j)} \in I^o : P^o \leftarrow \bigcup_{i,j} TC_i^{(j)}.P$;

```
/* Check schedulability condition and accept if possible. */
if schedule_type=non_preemptive then Do
    if ( result ← TC_in.P ⋈̂ P° ) then
            Already_Accepted ← Already_Accepted ∪ {TC_in} ;
    return(result) ;
oD ;
if schedule_type=preemptive then Do
    S_V ← S( I_V ← I° ∪ {TC_in} − {TC_z} ) ;
    if ( result ← ∀s_v ∈ S_V :
        (∀i : TC_i ∈ I_V : Σ_i Σ_{∀h:P_i^(h) ∈ P_i} ‖s_v ⊓ P_i^(h)‖ ≤ ‖s_v‖)
        ∧ (Σ ‖s_v‖ ≤ ‖v‖ − Σ_h ‖P_z^(h)‖) ) then
            Already_Accepted ← Already_Accepted ∪ {TC_in} ;
    return(result) ;
oD ;
□
```

4 Properties

We assume that each of the already accepted time constraints in the calendar have passed successfuly the same schedulability test. However, this property is valuable only if we show that time constraints that have been accepted after others, do not contradict this condition for the earlier. In order to show the above, we examine two possible cases that might happen when we accept a time constraint TC'' after TC' has already been accepted.

1. $TC'' \bowtie TC'$: TC' is not affected because there is no intersection between the two time constraints. The same argument is valid for both preemptive and non-preemptive policies.

2. $TC'' \not\bowtie TC'$: When checking the feasibility for TC'', a portion of $TC' \sqcap TC''$, out of TC', is assumed to consume its maximal resource requirements that it is capable to consume within the new verification window, which is the only domain that is affected. In the preemptive policy, when the schedulability criterion is examined, the time constraint accepted earlier is checked either as one member of the set of maximal convex subintervals or as the constraint that sets the verification window, TC_z.

$$\exists s_v \in S(I_V) : TC' \sqcap s_v \neq \phi \vee TC' = TC_z.$$

Hence, if schedulability is verified, then TC' is not affected. In the non-preemptive case, P' is verified to have sufficient laxity as a member of I^o, and therefore accepting P'' is possible only if the updated laxity of P' is positive.

4.1 Convergence of the Alogorithms

The algorithms presented in sections 3.3.1 and 3.3.2 contain "unbounded" loops of the form

$$\textbf{repeat} \cdots \textbf{until } I_4 \neq \phi.$$

In this section we show that the loop is bounded to at most two iterations. There are two cases we need to consider. The first is the case where the incoming time constraint TC_{in} is not contained within any other already accepted time constraint in the calendar. The second case is that where there exists such an already accepted time constraint.

1. $\nexists TC_i^{(j)} : TC_i^{(j)} \gg TC_{in}$: In that case v is initially set to TC_{in} and I_4 is initially set to ϕ. Since $\nexists TC_i^{(j)} : TC_i^{(j)} \gg TC_{in}$, I_4 is not updated in the classification switch, and hence only one iteration is executed.

2. $\exists TC_i^{(j)} : TC_i^{(j)} \gg TC_{in}$: In that case v is initially set to TC_{in} and I_4 is initially set to ϕ. Since $\exists TC_i^{(j)} : TC_i^{(j)} \gg TC_{in}$, at the end of the first iteration

   $$I_4 = \{TC_k'\}\ ,\ k = 1, \ldots$$

   such that

   $$\forall k : TC_k' \gg TC_{in}.$$

   We therefore set v to TC'_{max} such that

   $$\nexists TC_k' \in I_4 : TC_k' \gg TC'_{max}$$

   and start the second iteration with reseting I_4 to ϕ. If the second iteration ends with $I_4 \neq \phi$ we have a contradiction. To show it, say $I_4 \neq \phi$. Therefore,

   $$\exists TC' : TC' \gg TC'_{max}.$$

   But then TC' also satisfies

   $$TC' \gg TC_{in}$$

   which implies

   $$TC' \in \{TC_k'\}\ ,\ k = 1, \ldots$$

   that contradicts the definition of TC'_{max}.

4.2 Convex Interval Schedulability Guarantee

4.2.1 Correctness of Guarantee in Non-Preemptive Scheduling

In order to show that the guarantee provided by the non-preemptive schedulabiliy condition (Condition 1) for a convex time constraint, we show that two properties hold. First, consistency between possible schedule of the computation interval and the allowed window of occurrence is maintained. Second, an accepted constraint is not in conflict with the already accepted constraints, and a possible schedule of them all exists.

If Condition 1 holds, then by the definition of *laxity* ≥ 0 we can write

$$\forall P_i \in \mathcal{P}^o : P_i \uparrow TC_i \vee P_i \ll TC_i \vee P_i \downarrow TC_i \vee P_i = TC_i,$$

and

$$P_{in} \uparrow TC_{in} \vee P_{in} \ll TC_{in} \vee P_{in} \downarrow TC_{in} \vee P_{in} = TC_{in}.$$

Hence, executing the subintervals $\{P_i\}$ and P_{in} at a schedule which is exactly as the calendar through which the algorithm verified Condition 1 is not in conflict with any of the time constraint. Each execution interval is within the occurrence window allowed for it.

The assumption that all the already accepted time constraints have been accepted by this algorithm, implies that

$$\forall i,j : P_i, P_j \in \mathcal{P}^o \wedge i \neq j : P_i \bowtie P_j.$$

As shown in [2] and by the definitions in Appendix A, from a disjoint relation $P_{in} \bowtie P_i^o$, that is the positive result of the test in Condition 1, one can infer the existence of an *idle* interval T_z between the incoming computation P_{in} and each of the already accepted computations $P_{i_o} \in \mathcal{P}^o$. In other words

$$\exists T_z : P_{i_o} \| T_z \| P_{in} \vee P_{in} \| T_z \| P_{i_o},$$

with $\|T_z\| > 0$. The existence of such a T_z for computation P_{i_o}, whose corresponding window of occurrence TC_{i_o} satisfies $TC_{i_o} \in I_1^o$, implies the existence of another interval T_y, for computation P_i.

$$\exists T_y : P_i \| T_z \| P_{in} \vee P_{in} \| T_y \| P_i.$$

P_i's corresponding window of occurrence, TC_i, satisfies $TC_i \in I_1$. The existence of T_y is implied, due to the way I_1^o is constructed from I_1 by shifting P_i to the right to produce P_{i_o}, such that

$$T_z \downarrow T_y \vee T_z = T_y.$$

Similarly, the following holds for the right side of the verification interval. If Condition 1 holds, then each computation interval has at least one possibility to be scheduled without a conflict with its constraint. Furthermore, as in the left side of the verification algorithm, Condition 1 implies

$$\exists T_z : P_{i_o} \| T_z \| P_{in} \vee P_{in} \| T_z \| P_{i_o},$$

for computation interval P_{i_o}, whose corresponding window of occurrence TC_{i_o} satisfies $TC_{i_o} \in I_3^o$. Hence, by the way I_3^o is constructed from I_3, we can have a T_y

$$T_z \uparrow T_y \vee T_z = T_y$$

such that

$$\exists T_y : P_i \| T_z \| P_{in} \vee P_{in} \| T_y \| P_i,$$

for computation interval P_i, whose corresponding window of occurrence TC_i satisfies $TC_i \in I_3$.

4.2.2 Correctness of Guarantee in Preemptive Scheduling

Condition 2 constitutes a method to check the feasibility of a preemptive schedule for an incoming convex time constraint TC_{in}. We show here that if Condition 2 holds, then a feasible schedule exists.

The verification of schedulability has to be examined for two possible cases. The first is the case in which the verification window equals the interval of occurrence of the new incoming time constraint. The second is the case in which it is not equal.

1. $TC_z = TC_{in}$: From the first test in Condition 2 we know that $\forall s_v \in S(I_V)$ for those i's for which $s_v = \bigsqcup_i TC_i^o$

$$\|s_v\| \geq \sum_{\forall i(s_v)} \|P_i^o\|.$$

Let P_i be the convex computation interval of time constraint TC_i, that have been "modified" in the algorithm to the relevant intervals in I^o to P_i^o and TC_i^o respectively. Then, by summing the above we get

$$\sum_{\forall s_v \in S(I_V)} \|s_v\| \geq \sum_{\forall s_v \in S(I_V)} \sum_{\forall i(s_v)} \|P_i^o\| \geq \sum_{\forall s_v \in S(I_V)} \sum_{\forall i(s_v)} \|P_i \sqcap TC_{in}\|.$$

The right side of the above equation originates in the construction of I_1^o and I_3^o. Equality holds for I_2, as well as for members of I_1^o and I_3^o whose computation intervals were already within the verification window. For those whose computation intervals were "pushed" into the verification interval, for worst case simulation, inequality holds.

Hence, if the second test of Condition 2 holds for TC_{in}

$$\|TC_{in}\| = \|V\| \geq \sum_{\forall s_v \in S(I_V)} \|s_v\| + \|P_{in}\| \geq \sum_{\forall s_v \in S(I_V)} \sum_{\forall i(s_v)} \|P_i^o\| + \|P_{in}\|$$

and we can conclude by replacing $\sum_{\forall s_v \in S(I_V)} \sum_{i(s_v)}$ with $\sum_{\forall i: TC_i \in I^o}$ to get

$$\|TC_{in}\| \geq \sum_{\forall i: TC_i \in I^o} \|P_i \sqcap TC_{in}\| + \|P_{in}\|.$$

The above equation simply states that the verification interval is large enough to include all the required computations.

2. $TC_z \neq TC_{in}$: In this case TC_{in} is a member of I_V, and definitely a with a "contained" relation with respect to the verification window. Therefore, if the first test of Condition 2 holds, then $\forall s_v \in S(I_V)$ for those i's for which $s_v = \bigsqcup_{i(s_v)} TC_i^o$

$$\|s_v\| \geq \sum_{\forall i(s_v)} \|P_i^o\|.$$

In this case

$$TC_i^o \in I^o \vee TC_i^o = TC_{in}.$$

As in the first case, summing all the cases we get

$$\sum_{\forall s_v \in S(I_V)} \|s_v\| \geq \sum_{\forall s_v \in S(I_V)} \sum_{\forall i(s_v)} \|P_i^o\|$$

where one of these $P_{i(s_v)}$ is P_{in}, and the rest are members of I^o. So, we can write

$$\sum_{\forall s_v \in S(I_V)} \|s_v\| \geq \sum_{\forall s_v \in S(I_V)} \sum_{\forall i(s_v)} \|P_i^o\| + \|P_{in}\|.$$

As in the first case, the construction of I^o yields

$$\sum_{\forall s_v \in S(I_V)} \|s_v\| \geq \sum_{\forall s_v \in S(I_V)} \sum_{\forall i(s_v)} \|P_i^o\| + \|P_{in}\| \geq \sum_{\forall s_v \in S(I_V)} \sum_{\forall i: TC_i \in I^o - \{TC_z\}} \|P_i \sqcap TC_z\| + \|P_{in}\|.$$

Now, when the second test in Condition 2 holds, then

$$\|TC_z\| = \|V\| \geq \sum_{\forall s_v \in S(I_V)} \|s_v\| + \|P_z\|.$$

Using our results from above

$$\|TC_z\| = \|V\| \geq \sum_{\forall s_v \in S(I_V)} \sum_{\forall i: TC_i \in I^o - \{TC_z\}} \|P_i \sqcap TC_z\| + \|P_z\| + \|P_{in}\|.$$

4.3 Non-Convex Interval Schedulability Guarantee

The conditions for the feasibility of a schedule for non-convex computation intervals, either for a preemptive policy (Condition 4) or for a non-preemptive policy (Condition 3), are in a way generalizations of the conditions for the convex computation intervals.

4.3.1 Correctness of Guarantee in Non-Preemptive Scheduling

As in the convex case, the requirement for non-negative laxity in Condition 3 assures that no conflict exists between computation intervals and their corresponding occurrence windows. These assurance holds both for the incoming new time constraint and for already accepted constraints.

$$\forall P_i^{(j)} \in \mathcal{P}^o : \biguplus_j P_i^{(j)} \uparrow TC_i \vee \biguplus_j P_i^{(j)} \ll TC_i \vee \biguplus_j P_i^{(j)} \downarrow TC_i \vee \bigcup_j P_i^{(j)} = TC_i,$$

and

$$\biguplus_j P_{in}^{(j)} \uparrow TC_{in} \vee \biguplus_j P_{in}^{(j)} \ll TC_{in} \vee \biguplus_j P_{in}^{(j)} \downarrow TC_{in} \vee \biguplus_j P_{in}^{(j)} = TC_{in}.$$

Hence, each computation interval is within the occurrence window it is allowed to be.

The assumption that all the already accepted time constraints have been accepted by this algorithm, and the property of preservation of the algorithm, imply that

$$\forall i,j : P_i, P_j \in \mathcal{P}^o \wedge i \neq j : P_i \widehat{\bowtie} P_j.$$

As in the convex case, one can therefore infer the existence of an *idle* interval T_x between the incoming computation P_{in} and each of the already accepted computations $P_{i_o} \in \mathcal{P}^o$. In other words

$$\forall k,n : P_{i_o}^{(k)} \in P_{i_o} \wedge P_{in}^{(n)} \in P_{in} :$$

$$\exists T_x^{(k,n)} : P_{i_o}^{(k)} \| T_x^{(k,n)} \| P_{in}^{(n)} \vee P_{in}^{(n)} \| T_x^{(k,n)} \| P_{i_o}^{(k)},$$

with $\forall k,n : \|T_x^{(k,n)}\| > 0$.

For every P_{i_o} that corresponds to $TC_{i_o} \in I_2$ the existence of

$$\min_{k,n}(\|T_x^{(k,n)}\|) > 0$$

assures a feasible schedule. For every $TC_{i_o} \in I_1^o$, since every computation inteval of I_1^o is a shifted right version of a computation interval of I_1, one can infer the existence of $\{T_y^{(k,n)}\}$, such that

$$\forall k,n : T_x^{(k,n)} \downarrow T_y^{(k,n)} \vee T_x^{(k,n)} = T_y^{(k,n)}.$$

Thus, we can use this property to conclude $\|T_y^{(k,n)}\| \geq \|T_z^{(k,n)}\|$; that yields

$$\min_{k,n}(\|T_y^{(k,n)}\|) > 0$$

which assures a feasible schedule. Similarly, for every $TC_{i_o} \in I_S^o$

$$\forall k,n: \; T_z^{(k,n)} \uparrow T_y^{(k,n)} \vee T_z^{(k,n)} = T_y^{(k,n)}.$$

Since $\|T_y^{(k,n)}\| \geq \|T_z^{(k,n)}\|$, we conclude

$$\min_{k,n}(\|T_y^{(k,n)}\|) > 0$$

which assures a feasible schedule.

4.3.2 Correctness of Guarantee in Preemptive Scheduling

Condition 4 can be analyzed in the same way that Condition 2 has been analyzed above. Here we examine the same two cases as above.

1. $TC_z = TC_{in}$: From the first test in Condition 4 we know that $\forall s_v \in S(I_V)$ for those i's for which $s_v = \bigsqcup_i TC_i^o$

$$\|s_v\| \geq \sum_{\forall i(s_v)} \sum_{\forall k: P_i^{(k)} \in P_i^o} \|P_i^{(k)}\|.$$

Summing the above and considering the "unmodified" computation intervals $\{P_i\}$ we receive

$$\sum_{\forall s_v \in S(I_V)} \|s_v\| \geq \sum_{\forall s_v \in S(I_V)} \sum_{\forall i(s_v)} \sum_{\forall k: P_i^{(k)} \in P_i^o} \|P_i^k\| \geq \sum_{\forall s_v \in S(I_V)} \sum_{\forall i: TC_i \in I^o} \sum_{\forall n: P_i^{(n)} \in P_i} \|P_i^{(n)} \sqcap TC_{in}\|.$$

If the second test in Condition 4 holds, then

$$\|TC_{in}\| = \|V\| \geq \sum_{\forall s_v \in S(I_V)} \|s_v\| + \|P_{in}\| \geq \sum_{\forall s_v \in S(I_V)} \sum_{\forall i(s_v)} \sum_{\forall k: P_i^{(k)} \in P_i^o} \|P_i^{(k)}\| + \|P_{in}\|$$

and we can conclude by replacing $\sum_{\forall s_v \in S(I_V)} \sum_{i(s_v)}$ with $\sum_{\forall i: TC_i \in I^o}$ to get

$$\|TC_{in}\| \geq \sum_{\forall i: TC_i \in I^o} \sum_{\forall n: P_i^{(n)} \in P_i} \|P_i^n \sqcap TC_{in}\| + \|P_{in}\|.$$

2. $TC_x \neq TC_{in}$ **:** As in the above case, the first test in Condition 4 yields

$$\|s_v\| \geq \sum_{\forall i(s_v)} \|P_i^o\|.$$

Here

$$TC_i^o \in I^o \vee TC_i^o = TC_{in}.$$

Summing

$$\sum_{\forall s_v \in S(I_V)} \|s_v\| \geq \sum_{\forall s_v \in S(I_V)} \sum_{\forall i(s_v)} \sum_{\forall k: P_{i(s_v)}^{(k)} \in P_{i(s_v)}^o} \|P_{i(s_v)}^{(k)}\|$$

where one of these $P_{i(s_v)}^o$ is P_{in}, and the rest are members of I^o. Hence,

$$\sum_{\forall s_v \in S(I_V)} \|s_v\| \geq \sum_{\forall s_v \in S(I_V)} \sum_{\forall i(s_v): TC_i \in I^o} \sum_{\forall k: P_{i(s_v)}^{(k)} \in P_{i(s_v)}^o} \|P_{i(s_v)}^{(k)}\| + \|P_{in}\|.$$

As in the convex case, it can be extended to

$$\sum_{\forall s_v \in S(I_V)} \|s_v\| \geq \sum_{\forall s_v \in S(I_V)} \sum_{\forall i: TC_i \in I^o - \{TC_x\}} \sum_{\forall n: P_{i(s_v)}^{(n)} \in P_{i(s_v)}} \|P_i^{(n)} \sqcap TC_x\| + \|P_{in}\|.$$

From the second test in Condition 4

$$\|TC_x\| = \|V\| \geq \sum_{\forall s_v \in S(I_V)} \|s_v\| + \sum_{\forall k: P_x^{(k)} \in P_x} \|P_x^{(k)}\|.$$

Using our results from above

$$\|TC_x\| = \|V\| \geq \sum_{\forall s_v \in S(I_V)} \sum_{\forall i: TC_i \in I^o - \{TC_x\}} \sum_{\forall n: P_i^{(n)} \in P_{i(s_v)}} \|P_i^{(n)} \sqcap TC_x\|$$

$$+ \sum_{\forall k: P_x^{(k)} \in P_x} \|P_x^{(k)}\| + \sum_{\forall k: P_{in}^{(k)} \in P_{in}} \|P_{in}^{(k)}\|.$$

5 Concluding Remarks

In real-time systems, the scheduling and the resource management must be integrated in order to provide the support for guaranteeing that deadlines are to be met. Furthermore, the explicit expression of time must be the base of the decision taking processes, to prevent undesired dependencies and conflicts. The model we have presented here and in our previous works ([15,16]) strongly supports the above motivation.

In this paper we have shown a set of temporal relations that support both convex and non-convex time intervals. These relations support reasoning about a variety of temporal orderings, for continuous intervals as well as for intervals that contain gaps. Using these relations, we have formalized temporal properties that concern schedule feasibility. We have also introduces data structures called *calendars*, and algorithms that check and verify conditions for the feasibility of a guaranteed schedule. The correctness of the guarantee is discussed and analyzed. Although not included in this paper, it should be stated that an implementation of the above mechanisms (data structures and algorithms) have been carried out in a real-time operating system project in our university.

Having these mechanisms, each of the system's entities (in our case objects or resources) is explicitly related to present and future time. This fact enriches the reasoning for planning purposes, and enhances the reliability of satisfaction of the stringent timing constraints of accepted jobs.

A Interval Relations

A.1 Convex Interval Relations

In the following definitions let A and B be convex time intervals, such that

$$A =< t_\alpha^A, t_\beta^A >, B =< t_\alpha^B, t_\beta^B > .$$

A set of thirteen binary relations between convex intervals is proposed in [1]:

- *equal* $(A = B)$

$$t_\alpha^A = t_\alpha^B \wedge t_\beta^A = t_\beta^B,$$

- *preceed* $(A \prec B)$ and its inverse *succeed* $(A \succ B)$

$$t_\beta^A < t_\alpha^B,$$

- *meet* $(A \parallel B)$ and its inverse *met-by* $(B \parallel^u A)$

$$t_\beta^A = t_\alpha^B,$$

- *overlap* $(A \oslash B)$ and its inverse *overlapped-by* $(B \oslash^u A)$

$$t_\alpha^A < t_\alpha^B < t_\beta^A < t_\beta^B,$$

- *start* $(A \uparrow B)$ and its inverse *started-by* $(B \uparrow^u A)$

$$t_\alpha^B = t_\alpha^A < t_\beta^A < t_\beta^B,$$

- *during* $(A \ll B)$ and its inverse *contain* $(B \gg A)$

$$t_\alpha^B < t_\alpha^A < t_\beta^A < t_\beta^B,$$

- *end* $(A \downarrow B)$ and its inverse *ended-by* $(B \downarrow^u A)$

$$t_\alpha^B < t_\alpha^A < t_\beta^A = t_\beta^B.$$

A summary of these relations is given in Figure 3. The relation *disjoint* $(\bowtie)$ can therefore be expressed as

$$A \bowtie B \equiv (A \prec B) \vee (A \succ B),$$

and all the containment possibilities as

$$(A \uparrow B) \vee (A \ll B) \vee (A \downarrow B).$$

A.2 Non-Convex Interval Relations

In [10], ways in which non-convex interval relations are derived have been examined. First, a non-convex interval relation can be generated from a convex interval relation, by generalising the convex relations by means of *functors*. Additional new relations, which are not due to the above functors, as well as relations which are based solely on the first and last convex subintervals of the non-convex intervals, are enumerated.

Let C and D be non-convex intervals, and let $\{c_i\}$ and $\{d_i\}$ be their corresponding sets of maximal convex subintervals. Let $\mathcal{R}$ and $\mathcal{Q}$ be convex relations defined in section 2.3.2. The relation functors suggested in [10] are listed below.

- *mostly*: C mostly-$\mathcal{R}$ D if

$$\forall d_j \in D : \exists c_i \in C : c_i \mathcal{R} d_j.$$

There might be other subintervals in C but not in D.

- *always*: C always-$\mathcal{R}$ D if and only if C mostly-$\mathcal{R}$ D and D mostly-$\mathcal{R}^u$ C, where $\mathcal{R}^u$ is the converse relation to $\mathcal{R}$.

For example, C always-$\gg$ D (i.e. C always-contains D) if and only if
C mostly-$\gg$ D

$$\forall d_j \in D : \exists c_i \in C : c_i \gg d_j,$$

and D mostly-$\ll$ C

$$\forall c_i \in C : \exists d_j \in D : d_j \ll c_i.$$

- *partially*: C partially-$\mathcal{R}$ D if and only if

$$X = \{x_i\} = \{c_i, d_i : c_i \mathcal{R} d_i\} \neq \phi,$$

$$\{C - X\} \sqcap \{D - X\} = \phi.$$

Notice that all the elements $c_i \in C$ and $d_i \in D$ for which $c_i \mathcal{R} d_i$ does not hold, must be disjoint for C partially-$\mathcal{R}$ D to hold.

- *sometimes*: C sometimes-$\mathcal{R}$ D if and only if

$$X = \{x_i\} = \{c_i, d_i : c_i \mathcal{R} d_i\} \neq \phi.$$

- *disjunction*: C $\mathcal{R} \vee ... \vee \mathcal{Q}$ D if and only if

$$\forall c_i \in C, \forall d_j \in D : c_i \mathcal{R} d_j \vee ... \vee c_i \mathcal{Q} d_j \vee c_i \bowtie d_j.$$

Another functor we find important, one which is not defined in [9,10] is

- *totally*: C totally-$\mathcal{R}$ D if and only if

$$\forall d_j \in D : \forall c_i \in C : c_i \mathcal{R} d_j.$$

For example, C totally-$\prec$ D if and only if

$$\forall d_j \in D : \forall c_i \in C : c_i \prec d_j,$$

which means that the rightmost maximal convex interval of C precedes the leftmost maximal convex interval of D.

Another example is the non-convex intervals disjoint ($\widehat{\bowtie}$) relation, that can be expressed as "totally-$\bowtie$", meaning

$$\forall d_j \in D : \forall c_i \in C : c_i \bowtie d_j.$$

In addition to non-convex relations generated from generalisation of convex relations by means of the above functors, there are important relations that cannot be generated this way. One category of such relations consists of new relations, out of which we emphasise here the *bar* relation ([10]). This relation is needed for expressing predicates on convexity of intervals.

- *bar*: a relation between two non-convex intervals, denoted $C \leftrightarrow D$, whose *union* is convex.

$$C \leftrightarrow D \text{ if and only if } C \sqcup D = C \uplus D.$$

The relation is symmetric and commutative.

Another important set of relations is based on convex relations between the leftmost and rightmost maximal convex subintervals of non-convex intervals. Generalisation by means of functors cannot generate all these possible relations, because these relations are based on specialisation of particular subintervals, the leftmost and the rightmost ones. Let $\triangleleft C$ denote the leftmost maximal convex subinterval of non-convex interval C, and let $\triangleright C$ denote the rightmost maximal convex subinterval of non-convex interval C. Some relations based on these subintervals can be expressed by use of the functors. For example,

$$C \text{ totally-}\prec D \text{ if and only if } \triangleright C \prec \triangleleft D.$$

However, a large variety of relations cannot. For example,

- *meet*: non-convex interval C meets non-convex interval D, denoted $C \widehat{\parallel} D$, if and only if $\triangleright C \parallel \triangleleft D$.

Another example is the containment property. We have generalised above one possible containment with C always-$\gg$ D (i.e. C always-contains D), where each of the maximal convex subintervals of D is contained by a maximal convex subinterval of C. However, there are other containment properties. For example,

- *surround*: non-convex interval C surrounds non-convex interval D, denoted $C \widehat{\gg} D$, if and only if

$$\biguplus\{C\} \gg \biguplus\{D\}.$$

For allowing reasoning on subintervals, we therefore write

$$((\triangleleft C \prec \triangleleft D) \vee (\triangleleft C \oslash \triangleleft D) \vee (\triangleleft C \parallel \triangleleft D) \vee (\triangleleft C \ll \triangleleft D) \vee (\triangleleft C \downarrow \triangleleft D))$$

$$\bigwedge$$

$$((\triangleright C \succ \triangleright D) \vee (\triangleright C \oslash^{u} \triangleright D) \vee (\triangleright C \parallel^{u} \triangleright D) \vee (\triangleright C \gg \triangleright D) \vee (\triangleright C \uparrow \triangleright D)).$$

The expression is much simpler for the disjoint case

$$(C \widehat{\bowtie} D) \wedge (\triangleleft C \prec \triangleleft D) \wedge (\triangleright C \succ \triangleright D).$$

A subset of relations that are based on the leftmost and rightmost maximal convex subintervals, concerns the relative start and end of the non-convex intervals. In [10] the following relations are enumerated:

- *start-before*: $C \,\widehat{\uparrow <}\, D$ if

$$((\lhd C \prec \lhd D) \vee (\lhd C \oslash \lhd D) \vee (\lhd C \parallel \lhd D) \vee (\lhd C \ll \lhd D) \vee (\lhd C \downarrow \lhd D)).$$

- *start-after*: $C \uparrow > D$ if

$$((\lhd C \succ \lhd D) \vee (\lhd C \oslash^{u} \lhd D) \vee (\lhd C \parallel^{u} \lhd D) \vee (\lhd C \ll \lhd D) \vee (\lhd C \downarrow \lhd D)).$$

- *start-at*: $C \,\hat{\uparrow}\, D$ if

$$\lhd C \uparrow \lhd D.$$

- *end-at*: $C \,\hat{\downarrow}\, D$ if

$$\rhd C \downarrow \rhd D.$$

- *end-after*: $C \,\widehat{\downarrow >}\, D$ if

$$((\rhd C \succ \rhd D) \vee (\rhd C \oslash^{u} \rhd D) \vee (\rhd C \parallel^{u} \rhd D) \vee (\rhd C \ll \rhd D) \vee (\rhd C \downarrow \rhd D)).$$

- *end-before*: $C \,\widehat{\downarrow <}\, D$ if

$$((\rhd C \prec \rhd D) \vee (\rhd C \oslash \rhd D) \vee (\rhd C \parallel \rhd D) \vee (\rhd C \ll \rhd D) \vee (\rhd C \downarrow \rhd D)).$$

Finally, a very important calssification of the above relations can be derived according to properties of non-intersecting versus strictly intersecting. The disjoint properties are of extreme importance for the allocation which is to verify schedulability of objects. We have already described above the disjointly-surround relation. Another important disjoint relation is the following.

- *disjointly-overlap*: $C \,\widehat{\bowtie \oslash}\, D$ if and only if

$$(C \,\widehat{\bowtie}\, D) \wedge (\lhd C \prec \lhd D) \wedge (\rhd C \prec \rhd D) \wedge (\lhd D \prec \rhd C).$$

If C is the set of already guaranteed services at a resource, and D is a new request for a service at this resource, then

$$((C \,\widehat{\bowtie}\, D) \wedge (C \,\widehat{\succ}\, D)) \vee (C \,\widehat{\bowtie \oslash}\, D)$$

can be a proper schedulability condition.

References

[1] Allen J., *Maintaining Knowledge about Temporal Intervals*, Communications of the ACM, Vol 26 No 11 pp 832-843, November 1983.

[2] Allen J. and Hayes P., *A Common Sense Theory of Time*, Proceedings of the Nineth International Joint Conference on Artificial Intelegenece (IJCAI), pp 528-531, August 18-23, 1985, Los Angeles, California.

[3] Booch G., *Object-Oriented Development*, IEEE Trans. on Software Engineering, Vol SE-12 No 2 pp 211-221, Feb 1986.

[4] Caspi P., Halbwachs N., *A Functional Model for Describing and Reasoning Time Behavior of Computer Systems*, Acta Informatica, Vol 22 No 6 pp 595-628, March 1986.

[5] Cheng S., Stankovic J. and Ramamrithan K., *Dynamic Scheduling of Groups of Tasks with Precedence Constraints in Distributed Hard Real-Time Systems*, Proccedings of Real-Time Systems Symposium (IEEE), pp 166-174, December 2-4, 1986, New Orleans, LA.

[6] Gora W., Herzog U. and Tripathi S., *Clock Synchronization on the Factory Floor*, Proc. of Workshop on Factory Communication, NBS, March 1987.

[7] Gusella R. and Zatti S., *The Accuracy of Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD*, technical report, Computer Science Division, University of California, Berkeley CA, December 1986.

[8] Harel D. and Pnueli A., *On The Development of Reactive Sysytems*, Weitzman Institute of Science, Rehovot, Israel, 1985.

[9] Ladkin P., *Primitives and Units for Time Specification*, Proceedings of the Fifth National Conference on Artificial Intelegenece (AAAI), pp 354-359, August 11-15, 1986, Philadelphia, PA.

[10] Ladkin P., *Time Representation: A Taxonomy of Interval Relations*, Proceedings of the Fifth National Conference on Artificial Intelegenece (AAAI), pp 360-366, August 11-15, 1986, Philadelphia, PA.

[11] Lamport L., *Time, Clocks and Ordering of Events in a Distributed System*, Communications of the ACM, Vol 21 No 7 pp 558-565, July 1978.

[12] Lamport L. and Melliar-Smith P. M., *Synchronizing Clocks in the Presence of Faults*, Journal of the ACM, Vol 32 No 1 pp 52-78, January 1985.

[13] Lamport L., *Synchronizing Time Servers*, SRC report No 18, DEC SRC, Palo Alto, CA, June 1987.

[14] Leban B., McDonald D. and Forster D., *A Representation for Collections of Temporal Intervals*, Proceedings of the Fifth National Conference on Artificial Intelegenece (AAAI), pp 367-371, August 11-15, 1986, Philadelphia, PA.

[15] Shem-Tov Levi and Ashok K. Agrawala, *Objects Architecture for Real-Time, Distributed, Fault Tolerant Operating Systems*, IEEE Workshop on Real-Time Operating Systems, Cambridge MA, July 1987.

[16] Shem-Tov Levi and Ashok K. Agrawala, *Objects Architecture: A Comprehensive Design Approach for Real-Time, Distributed, Fault-Tolerant, Reactive Operating Systems*, Technical Report CS-TR-1915, Department of Computer Science, University of Maryland, College Park, Maryland, September 1987.

[17] Marzullo K. and Owicki S., *Maintaining the Time in a Distributed System*, ACM Operating Systems Review, Vol 19 No 3 pp 44-54, July 1985.

[18] Mok A. K. and Dertousos M. L., *Multiprocessor Scheduling in A Hard Real-Time Environment*, Proccedings of the Seventh Texas Conference on Computing Systems, pp 5.1-5.12, October 30 - November 1, 1978, Houston, Texas.

[19] Rit J., *Propagating Temporal Constraints for Scheduling*, Proceedings of the Fifth National Conference on Artificial Intelegenece (AAAI), pp 383-388, August 11-15, 1986, Philadelphia, PA.

[20] Schwan K., Bo W. and Gopinath P., *A High Performence, Object Based Operating System for Real-Time, Robotics Applications*, Proccedings of Real-Time System Symposium (IEEE), pp 147-156, New Orleans, Louisiana, December 2-4, 1986.

[21] Schwan K., Bihari T. and Weide B., *High Performence Operating System Primitives for Robotics and Real-Time Control Systems*, ACM Trans. on Computer Systems, Vol 5 No 3 pp 189-231, August 1987.

[22] Shankar A. U. and Lam S. S., *Time-Dependent Distributed Sytems: Proving Safety, Liveness and Real-Time Properties*, technical report CS–TR–1586, Department of Computer Science, University of Maryland, College Park MD, Dec 1985.

[23] Tripathi S. and Chang S., *A Clock Synchronization Algorithm for Hierarchical LANs – Implementation and Measurements*, Technical Report TR-86-48, Systems Research Center, University of Maryland, College Park, Maryland, 1986.

[24] Vilain M. and Kauts H., *Constraint Propagation Algorithms for Temporal Reasoning*, Proceedings of the Fifth National Conference on Artificial Intelegenece (AAAI), pp 377-382, August 11-15, 1986, Philadelphia, PA.

[25] Volts R. A. and Mudge T. N., *Instruction Level Timing Mechanism for Accurate Real-Time Task Scheduling*, IEEE Trans on Computers, Vol C-36 No 8 pp 988-993, August, 1987.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CS-TR-1954	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION University of Maryland 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research
6c. ADDRESS (City, State, and ZIP Code) Dept. of Computor Science University of Maryland College Park, MD 20742	7b. ADDRESS (City, State, and ZIP Code) 800 North Quincy St. Arlington, VA 22217-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-87-K-0241
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

Temporal Relations and Structures in Real-Time Operating Systems

12. PERSONAL AUTHOR(S)

Shem-Tov Levi and Ashok K. Agrawala

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	1987 December 16	48

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The temporal properties of objects in a real-time, distributed, fault-tolerant, reactive operating systems are defined and analyzed. Accordingly, properties associated with the schedulability of accepted jobs whose deadlines are guaranteed are examined. Special mechanisms that support temporal inference are proposed. These mechanisms support explicit time expression, precedence relations, and projections of the knowledge of real-time at different localities. Special data structures, called *calendars*, are proposed for management and planning of activities and for scheduling. Algorithms that verify schedulability of arriving requests are introduced, ensuring that already-given guarantees for already-accepted jobs are not violated.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

END

DATE

FILMED

6-1988

DTIC