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
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
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.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited
Contents

1 Introduction
   1.1 Objects Architecture: A Review of Previous Work ................................... 3
   1.2 Time-Service in Hard Real-Time Systems .................................................. 6
   1.3 Guarantees in Hard Real-Time Systems ..................................................... 8

2 Notations
   2.1 Time Representation .................................................................................. 10
   2.2 Intervals Versus Points ............................................................................. 10
   2.3 Temporal Relations ................................................................................... 11
      2.3.1 Basic Definitions ............................................................................... 11
      2.3.2 Convex Interval Relations ............................................................... 12
      2.3.3 Non-Convex Interval Relations ......................................................... 13

3 Schedulability
   3.1 Constraints Propagation ........................................................................... 16
   3.2 Allocation and Scheduling Orientation ..................................................... 19
      3.2.1 Convex Time Constraints ................................................................. 20
      3.2.2 Non-convex Time Constraints ......................................................... 24
   3.3 Scheduling Feasibility Verification Algorithms ......................................... 27
      3.3.1 Convex Constraint Allocation Algorithm ......................................... 29
      3.3.2 Non-Convex Constraint Allocation Algorithm ................................... 32

4 Properties
   4.1 Convergence of the Algorithms ................................................................ 35
   4.2 Convex Interval Schedulability Guarantee ................................................ 36
      4.2.1 Correctness of Guarantee in Non-Preemptive Scheduling .................... 36
      4.2.2 Correctness of Guarantee in Preemptive Scheduling ........................... 37
   4.3 Non-Convex Interval Schedulability Guarantee ......................................... 39
      4.3.1 Correctness of Guarantee in Non-Preemptive Scheduling .................... 39
      4.3.2 Correctness of Guarantee in Preemptive Scheduling ........................... 40

5 Concluding Remarks ....................................................................................... 41

A Interval Relations
   A.1 Convex Interval Relations ......................................................................... 43
   A.2 Non-Convex Interval Relations .................................................................. 44

2

DTC
Copy

Availabilities
Codes

Dist Avail and/or
Special

A-1
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 hard¹ 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

---

¹Hard real-time systems are characterized 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
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 \( T_p(t) \), 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)
  \[
a_p(t) = 1 + \frac{C_p^{(i)}(t_p^{(i)}) - T_p(t_p^{(i)})}{J}
  \]
  \[
b_p(t) = b_p(t_p^{(i-1)}) + (a_p(t_p^{(i-1)}) - a_p(t_p^{(i)})) C_p^{(i)}(t_p^{(i)}).
\]
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 \( Tbef(T_2) \) is true for \( T_i(t) \leq T_2 \) 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 occurred. 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 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 occurred 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-
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

\[ < \text{Id}, \text{Taft}(\text{condition}_1), c_{Id}, f_{Id}, \text{Tbef}(\text{condition}_2) > \]

where:

- \text{Id} is the name of the executable object (process) in the proper context,
- \text{Taft}(\text{condition}_1) states after what event should execution begin. Simple \textit{true} stands for "as soon as possible”,
- \( c_{Id} \) is the computation time of object \text{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 occurrence of execution,
- \( \text{Tbef}(\text{condition}_2) \) states the deadline \( d_{Id} = T_d - \text{Now} \) which should be met. \textit{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 \text{Taft}(\text{condition}_1) and \text{Tbef}(\text{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 support 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 \( t_a \) and \( t_\beta \) be such time points.

Definition 2 Two binary relations are defined for time points.

- \( t_a < t_\beta \) (\( t_a \) before \( t_\beta \)) and its inverse \( t_a > t_\beta \) (\( t_a \) after \( t_\beta \)).
- \( t_a = t_\beta \). \)

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

\[
< t_a, t_\beta > = \{ t : t_a \leq t \leq t_\beta \}. \)

The above definition implies that a time point \( t_a \) can be associated with the time interval \( < t_a, t_a > \).

Definition 4 The duration of a convex time interval \( < t_a, t_\beta > \) is a measure of the interval such that

\[
\| < t_a, t_\beta > \| = |t_\beta - t_a|. \)

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.
A = B  equal  \[ \begin{array}{c}
| A | \\
| B |
\end{array} \]
A < B  preceed  succeed  \[ B > A \]
A || B  meet  met-by  \[ B \parallel A \]
A \& B  overlap  overlapped-by  \[ B \& A \]
A ↑ B  start  started-by  \[ B ↑ A \]
A < B  during  contain  \[ B > A \]
A ↓ B  end  ended-by  \[ B ↓ A \]

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*,
- *preceed* 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

\[
A \bowtie B \equiv (A < B) \lor (A > B),
\]
and all the containment possibilities as

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

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

$$A = \langle t^A_\alpha, t^A_\beta \rangle, B = \langle t^B_\alpha, t^B_\beta \rangle.$$

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 $\neg(A < B) \land \neg(A > B)$ as

$$A \cap B \equiv \langle \max(t^A_\alpha, t^B_\alpha), \min(t^A_\beta, t^B_\beta) \rangle$$

and is defined $\phi$ (null) for $(A < B) \lor (A > B)$. $\Box$

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

$$A \cup B \equiv \langle \min(t^A_\alpha, t^B_\alpha), \max(t^A_\beta, t^B_\beta) \rangle. \Box$$

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

\[
\{C\} \cup \{D\} \equiv S(\{C\}) \cup S(\{D\}). \tag{\text{□}}
\]

Note that the cover of the set union

\[
\bigcup (\{C\} \cup \{D\}) = < \min(t^C_a, t^C_b), \max(t^D_a, t^D_b) >.
\]

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

\[
\bigcup_{i=1}^n \{c_i\} = c_1 \cup c_2 \cup c_3 \cup \ldots \cup c_n.
\]

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

\[
E = C \cup D,
\]

such that

\[
E = S(\{C\} \cup \{D\}). \tag{\text{□}}
\]

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

\[
\bigcup_{i=1}^n \{c_i\} = c_1 \cup c_2 \cup c_3 \cup \ldots \cup c_n.
\]

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 \( R \) and \( Q \) be convex relations defined in section 2.3.2.

- **mostly:** \( C \) mostly-\( R \) \( D \) if

\[
\forall d_j \in D : \exists c_i \in C : c_i R d_j.
\]
• **always:** \( C \always R D \) if and only if \( C \mostly R D \) and \( D \mostly R^c \), where \( R^c \) is the converse relation to \( R \).

• **partially:** \( C \partially R D \) if and only if

\[
X = \{ x_i \} = \{ c_i, d_i : c_i R d_i \} \neq \phi, \\
\{ C - X \} \cap \{ D - X \} = \phi.
\]

• **sometimes:** \( C \sometimes R D \) if and only if

\[
X = \{ x_i \} = \{ c_i, d_i : c_i R d_i \} \neq \phi.
\]

• **disjunction:** \( C R \lor ... \lor Q D \) if and only if

\[
\forall c_i \in C, \forall d_j \in D : c_i R d_j \lor ... \lor c_i Q d_j \lor c_i \bowtie d_j.
\]

• **totally:** \( C \totally R D \) if and only if

\[
\forall d_j \in D : \forall c_i \in C : c_i R d_j.
\]

For example, the non-convex intervals disjoint (\( \bowtie \)) 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 \(<C \) and the rightmost one \( >C \).

### 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
the second and third phases into one. We find it very expensive to choose computation localities upon each context switching, and maintain three phases. 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

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

The conditions imposed on beginning, $Taft(condition_1)$, and on end, $Tbef(condition_2)$, create a time window whose duration must be at least $c_{\Delta}$. 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.

---

*See section 1.3.*
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 $(\text{begin}, \text{end})$ point coordinates. Projecting the latest possible $\text{end}(TC_1)$ onto the $\text{begin}$ axis of $TC_2$, creates a window of occurrences, denoted $A$, whose points stand for occurrences for which the relation $TC_1 < TC_2$ definitely holds. Projecting the earliest possible $\text{begin}(TC_1)$ onto the $\text{end}$ axis of $TC_2$, creates the section $B$, for which $TC_2 < TC_1$ definitely holds. Section $C$ is characterized with

\[
(\text{begin}(TC_2) < \text{end}_{\min}(TC_1)) \land (\text{end}(TC_2) > \text{end}_{\max}(TC_1)).
\]

In other words, $TC_1 \cap TC_2$ is definitely non-null in section $C$. The two streaked areas in Figure 5 show regions in which the temporal relation between the time constraints is not certain. In these regions the relation can be determined only upon an accurate knowledge of the actual $(\text{begin}, \text{end})$ 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 $R_{i,j}$ is required
between \( TC_i \) and \( TC_j \), then \( TC_i \cap TC_j \) is achieved if we intersect \( TC_j \) with the region allowed by \( TC_i \) and \( R_{i,j} \). 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 \texttt{get-time} request is

\[
T_p(t) = a_p(t)C_p(t) + b_p(t), \quad t \geq t_p^{(0)}
\]

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

\[
\begin{align*}
\text{begin}_{\text{min}}'(TC_i) &= \text{begin}_{\text{min}}(TC_i) + \delta a_p \text{begin}_{\text{min}}(TC_i) + \delta b_p, \\
\text{begin}_{\text{max}}'(TC_i) &= \text{begin}_{\text{max}}(TC_i) - \delta a_p \text{begin}_{\text{max}}(TC_i) - \delta b_p, \\
\text{end}_{\text{min}}'(TC_i) &= \text{end}_{\text{min}}(TC_i) + \Delta a_p \text{end}_{\text{min}}(TC_i) + \Delta b_p, \\
\text{end}_{\text{max}}'(TC_i) &= \text{end}_{\text{max}}(TC_i) - \Delta a_p \text{end}_{\text{max}}(TC_i) - \Delta b_p.
\end{align*}
\]

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

\[(P_i^{(j)} \uparrow TC_i^{(j)}) \lor (P_i^{(j)} \ll TC_i^{(j)}) \lor (P_i^{(j)} \downarrow TC_i^{(j)}).\]

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

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

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

\[
\text{backward.slack} = x_- = t_a^p - t_a^{TC},
\]

\[
\text{forward.slack} = x_+ = t_b^{TC} - t_b^p.
\]

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

---

4Begin filter.

5Deadline.
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:

$$TC_{in} = \langle \text{begin}_{\min}(TC_{in}), \text{end}_{\max}(TC_{in}) \rangle.$$  

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**

$$V = \langle t_a^V, t_d^V \rangle.$$  

Initially, $V = TC_{in} = \langle \text{begin}_{\min}(TC_{in}), \text{end}_{\max}(TC_{in}) \rangle$. Each subinterval that intersects with $V$, denoted $TC_i^{(j)}$, has one of four possible relations with it.

1. $TC_i^{(j)} \cap V$, or
2. $(TC_i^{(j)} \uparrow V) \lor (TC_i^{(j)} \prec V) \lor (TC_i^{(j)} \downarrow V)$, or
3. $(TC_i^{(j)} \cup^u V$, or
4. $(TC_i^{(j)} \uparrow^u V) \lor (TC_i^{(j)} \rightarrow V) \lor (TC_i^{(j)} \downarrow^u V)$.

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 $V$, $I_2$ the subset of those that are overlapped by $V$, $I_3$ 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

$$TC_i^{(j)} \in I_2 : P_i^{(j)} \uparrow TC_i^{(j)}$$

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 $I_1^*$ and $I_2^*$, for which only the relevant portions of overlapping time constraints are accounted for.

$$I_1^* \equiv \{ TC_i^{(j)} \} = \{ \langle \max(\begin{array}{cc} \text{begin}_{\text{min}}(TC_i^{(j)}), t'_a \end{array}), end_{\text{max}}(TC_i^{(j)}) \rangle, TC_i^{(j)} \in I_1 \}$$

$$I_2^* \equiv \{ TC_i^{(j)} \} = \{ \langle \begin{array}{cc} \text{begin}_{\text{min}}(TC_i^{(j)}), \min(t'_a, end_{\text{max}}(TC_i^{(j)})) \end{array} \rangle, TC_i^{(j)} \in I_2 \}$$

For each of the subintervals in $I_1^*$ and $I_2^*$ the computation requirement $P_i^{(j)}$ is appropriately set to

$$\|P_i^{(j)}\| = \min(\|P_i^{(j)}\|, \|TC_i^{(j)}\|).$$

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

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

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

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

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

$$\{ P_{in} \} \propto P^*$$

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

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 $TC_1$, $I_2$ contains one convex subinterval of $TC_2$ 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 ($\propto$) 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 $I^*$ that have either overlapping or containment relations with $TC_{in}$, affect its laxity. However, their own laxities are affected.
by \( TC_{in} \) as well. Figure 7 demonstrates a case of two overlapping constraints, \( TC_A \cap TC_B \), 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 \( z_{-}^B \) is negative, and \( P_B \) can be moved to the right (on \( z_{+}^B \)'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,

\[
x_{+}^A = \min( x_{+}^A, \min(t_{\beta}^{TC_A} - t_{\beta}^{P_A}, (t_{\beta}^{P_B} + x_{+}^B) - t_{\beta}^{P_A}))
\]

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.
If the context switching can be regarded as negligible, then one can allocate a resource portion \((\gamma_A, \gamma_B)\) per time unit during the time intervals \(TCA\) and \(TCB\) respectively, such that

\[
\gamma_A = \frac{\|P_A\|}{\|TCA\|} \leq 1
\]

\[
\gamma_B = \frac{\|P_B\|}{\|TCB\|} \leq 1
\]

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

\[
\int_{TCA} \gamma_A \, dt = P_A
\]

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 exceeds 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 reasonable 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

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

Recalling the definition of the set of maximal convex subintervals\(^*\), 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 \( TC_{in} \), for which \( TC_a = TC_{in} \) or \( TC_a \gg TC_{in} \). Let \( I_V \) be

\[
I_V = I^* \cup \{TC_{in}\} - \{TC_a\}
\]

where \( I^* \) is the set union of \( I^*_1, I^*_2 \) and \( I^*_3 \).

\( TC_{in} \) is preemptively schedulable if

\[
\forall s_v \in S(I_V): \sum_{V_i: TC_i \in I_V} \|s_v \cap P_i\| \leq \|s_v\| \wedge \sum_{V_v \in S(I_V)} \|s_v\| \leq \|V\| - \|P_a\|
\]

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

\(^*\)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

$$((\bigcup_j P_i^{(j)} \uparrow TC_i) \lor (\bigcup_j P_i^{(j)} \leftarrow TC_i) \lor (\bigcup_j P_i^{(j)} \downarrow TC_i)).$$

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 $(x_i^{TC_i}, x_i^{TC_i})$, such that

$$x_i^{TC_i} \leq t_{\alpha}^{P_i} - t_{\alpha}^{TC_i} = t_{\alpha}^{P_i} - t_{\alpha}^{TC_i},$$

$$x_i^{TC_i} \leq t_{\beta}^{TC_i} - t_{\beta}^{P_i} = t_{\beta}^{TC_i} - t_{\beta}^{P_i},$$

where $P_i^L = \langle P_i$ and $P_i^R = \rangle P_i$. □.

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 $\bigcup_j P_i$, and $\bigcup_j P_i$ 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 $P_i$.

$$\forall TC_i \in I_1 : P_i \uparrow \{TC_i\}$$
setting $x_{+}^{TC_{i}} \leftarrow 0$, and
\[ \forall TC_{i} \in I_{3} : P_{i}^{\hat{\text{C}}} \{ TC_{i} \} \]
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.

\[ I_{1}^* = \{ TC_{i} \} = \{ < \max(\text{begin}_{\min}(TC_{i}), t_{a}^{V}), \text{end}_{\max}(TC_{i}) >, TC_{i} \in I_{1} \} \]

\[ I_{3}^* = \{ TC_{i} \} = \{ < \text{begin}_{\min}(TC_{i}), \min(t_{p}^{V}, \text{end}_{\max}(TC_{i})) >, TC_{i} \in I_{3} \} \]

For each of the subintervals in $I_{1}^*$ and $I_{3}^*$ the non-convex computation requirement $P_{i}$ is appropriately set to
\[ P_{i} = \{ \bigcup_{j} P_{i}^{(j)} | P_{i}^{(j)} = P_{i}^{(j)} \cap TC_{i}, P_{i}^{(j)} \in P_{i} \}. \]

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-$\rightarrow$ 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
\[ \forall i \forall j : TC_{i}^{(j)} \in I^* : \exists x_{-}^{TC_{i}^{(j)}} \geq 0. \exists x_{+}^{TC_{i}^{(j)}} \geq 0. \exists x_{-}^{TC_{in}} \geq 0. \exists x_{+}^{TC_{in}} \geq 0. : \]
\[ \forall P_{i}^{(j)} \in P^* : P_{in} \cap P_{i}^{(j)} \]
where $I^*$ is the set union of $I_{1}^*$, $I_{2}$ and $I_{3}^*$ and $P^*$ is the set $\{ P_{i}^{(j)} | TC_{i}^{(j)} \in I^* \}$.

In the preemptive 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 TCz, for which TCz = TCin or TCz > TCin. Let 
Iv 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

\[ \forall s_o \in S(I_v) : \sum_{\forall i:TC_{in} \in I_v} \sum_{\forall k:P_{i}^{(k)} \in P_i} ||s_o \cap P_{i}^{(k)}|| \leq ||s_o|| \]

\[ \land \sum_{\forall s_o \in S(I_v)} ||s_o|| \leq ||V|| - \sum_{\forall k:P_{s}^{(k)} \in P_s} ||P_{s}^{(k)}|| \]

where s_o are the convex subintervals of S(I_v). \[ \square \]

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 exclusion, to avoid concurrent accesses and false deduction. The PUSHTC 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 timeconstraint = construct
    { tc: convex.time.interval;
      back.slack, forward.slack: real;
      P: convex.time.interval;
      freq: real;
      state: integer };
global var Already.Accepted: set of time_constraints;

boolean function scheduler.PUSH.TC (TCin:time.constraint, schedule_type: class);

local var J1, J2, I3, I4, P1, P2, I5: set of time.constraint;
    TC(j), TCin, constraint: time_constraint;
    S: set of convex.time.interval;
    v, s: convex.time.interval;
    i, j, i0, j0, k, n: indices;

/* Init */
TCn ← TCin;
v ← TCin.tc;
Jn ← φ;

/* Prepare verification interval and classify interference set
of already accepted time constraints. */
repeat
  I1, I2, I3, I4, Pu, P2u, I5u ← φ;
  i ← 0;
  ∀constraint ∈ Already.Accepted Do
    if constraint.freq ≠ 0. then Do
      j ← [(v.begin − constraint.tc.begin)/constraint.freq];
      TC(j).tc.begin ← constraint.tc.begin + j/constraint.freq;
      TC(j).tc.end ← constraint.tc.end + j/constraint.freq;
      oD;
    else Do
      j ← 0;
      TC(j) ← constraint;
      oD;
  od;}
while $T C (i).t c .b e g i n < v .e n d$ Do

$$\begin{align*}
&T C (i).t c \Theta v \rightarrow I_1 \leftarrow I_1 \cup \{T C (i)\}; \\
&(T C (i).t c \uparrow v) \vee (T C (i).t c \leftarrow v) \vee (T C (i).t c \downarrow v) \rightarrow I_2 \leftarrow I_2 \cup \{T C (i)\}; \\
&T C (i).t c \Theta^* v \rightarrow I_3 \leftarrow I_3 \cup \{T C (i)\}; \\
&(T C (i).t c \uparrow v) \vee (T C (i).t c \Rightarrow v) \vee (T C (i).t c \downarrow v) \rightarrow I_4 \leftarrow I_4 \cup \{T C (i)\}; \\
\end{align*}$$

/* Next instance of periodic constraints */
if $c o n s t r a i n t . f r e q \neq 0$, then Do

$$j \leftarrow j + 1;$$

$$T C (i).t c .b e g i n \leftarrow c o n s t r a i n t .t c .b e g i n + j / c o n s t r a i n t . f r e q ;$$

$$T C (i).t c .e n d \leftarrow c o n s t r a i n t .t c .e n d + j / c o n s t r a i n t . f r e q ;$$

else

break;

oD;

i \leftarrow i + 1;

oD;

if $I_4 \neq \emptyset$ then Do

$$T C_4 \leftarrow m a x . c o n v e x . c o n s t r a i n t (I_4);$$

$$v \leftarrow m a x . c o n v e x . i n t e r v a l (I_4);$$

oD;

until $I_4 = \emptyset$;

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

$$\forall V j : T C (i).t c (I_1) \in I_1 : D o$$

$$T C (3x).t c .b e g i n \leftarrow m a x (v .b e g i n , T C (i).t c .b e g i n );$$

$$T C (3x).t c .e n d \leftarrow T C (i).t c .e n d ;$$

$$R_i \leftarrow R_i \cup \{T C (3x)\};$$

oD;

$$\forall V j : T C (i).t c (I_3) \in I_3 : D o$$

$$T C (3x).t c .e n d \leftarrow m i n (T C (i).t c .e n d , v .e n d );$$

$$T C (3x).t c .b e g i n \leftarrow T C (i).t c .b e g i n ;$$

$$I_3 \leftarrow I_3 \cup \{T C (3x)\};$$

oD;

$$P \leftarrow \{T C (i) : (T C (i) \in R_i) \vee (T C (i) \in I_2) \vee (T C (i) \in I_3) \};$$
/ * Check schedulability condition and accept if possible. */

if schedule.type=non_preemptive then Do
  if (result ← ∀∀j : TC_j \( j \) ∈ P :
    \( \exists TC_j \cdot \text{back.slack} \geq 0 \land \exists TC_j \cdot \text{for.slack} \geq 0 \land \exists TC_j \cdot \text{back.slack} \geq 0 \land \exists TC_j \cdot \text{for.slack} \geq 0 \) :
    TC_j \cdot P = TC_j : P \) then
    Already.Accepted ← Already.Accepted ∪ {TC_j} ;
  return(result) ;
endif

if schedule.type=preemptive then Do
  S_v ← S(IV ← P ∪ {TC_v} \{TC_u}\) ;
  if (result ← ∀s : S_v :
    (\( ∀i : TC_i \in IV : \sum \|P_i \cap s_v\| \leq \|s_v\|\)
    ∧ (\( \sum \|s_v\| \leq \|u\| - \|TC_u \cdot P\|\) \) then
    Already.Accepted ← Already.Accepted ∪ {TC_v} ;
  return(result) ;
dendif
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 (TCin:time_constraint, schedule_type: class);
local var P^, I_1, I_2, I_3, I_4, P_1, P_2, I_V: set of time_constraint ;
P*: set of non_convex_time_interval ;
TC_1, TC_V: set of time_constraint ;
I_1, I_V: set of convex_time_interval ;
i, j, i, j, k, n: indices ;

/* Init */
TC_1 ← TC_in ;
I_1 ← TC_in.begin ;
P^* ← ∅ ;

/* Prepare verification interval and classify interference set of already accepted time constraints. */
repeat
I_1, I_2, I_3, I_4, P_1, P_2 ← ∅ ;
i ← 0 ;
Vconstraint ∈ Already_Accepted Do
  if constraint.freq ≠ 0. then Do
    j ← [(u.begin - constraint.tc.end)/constraint.freq] ;
    TC_1.begin ← constraint.tc.begin + j/constraint.freq ;
    TC_1.end ← constraint.tc.end + j/constraint.freq ;
    oD ;
  else Do
    j ← 0 ;
    TC_1 ← constraint ;
    oD ;
  end Do ;
/* Classification switch */

while TC_{i}^{(1)} .tc.begin < v.end Do

\[ TC_{i}^{(1)} .tc \cap v \rightarrow I_{1} \leftarrow I_{1} \cup \{ TC_{i}^{(1)} \}; \]
\[ (TC_{i}^{(1)} .tc \uparrow v) \lor (TC_{i}^{(1)} .tc \ll v) \lor (TC_{i}^{(1)} .tc \downarrow v) \rightarrow I_{2} \leftarrow I_{2} \cup \{ TC_{i}^{(1)} \}; \]
\[ TC_{i}^{(1)} .tc \cap v \rightarrow I_{3} \leftarrow I_{3} \cup \{ TC_{i}^{(1)} \}; \]
\[ (TC_{i}^{(1)} .tc \uparrow v) \lor (TC_{i}^{(1)} .tc \gg v) \lor (TC_{i}^{(1)} .tc \downarrow v) \rightarrow I_{4} \leftarrow I_{4} \cup \{ TC_{i}^{(1)} \}; \]

/* Next instance of periodic constraints */

if constraint.freq \neq 0. then Do

\[ j \leftarrow j + 1; \]
\[ TC_{i}^{(1)} .tc.begin \leftarrow \text{constraint.tc.begin} + j / \text{constraint.freq}; \]
\[ TC_{i}^{(1)} .tc.end \leftarrow \text{constraint.tc.end} + j / \text{constraint.freq}; \]

oD

else

break;

oD;

i \leftarrow i + 1;

oD;

if I_{4} \neq \emptyset then Do

\[ TC_{i} \leftarrow \max .\text{convex.constraint}(I_{4}); \]
\[ v \leftarrow \max .\text{convex.interval}(I_{4}); \]

oD;

until I_{4} = \emptyset;

/* Update the overlapping already-accepted constraints \( (I_{1}, I_{3}) \) to contain only relevant portion within the verification interval. */

\[ \forall \forall j : TC_{i}^{(1)} \in I_{1} : \text{Do} \]
\[ TC_{i}^{(1)} .tc.begin \leftarrow \max (v.begin, TC_{i}^{(1)} .tc.begin); \]
\[ TC_{i}^{(1)} .tc.end \leftarrow TC_{i}^{(1)} .tc.end; \]
\[ TC_{i}^{(1)} .tc.begin \leftarrow \text{constraint.tc.begin}; \]
\[ TC_{i}^{(1)} .tc.end \leftarrow \text{constraint.tc.end}; \]
\[ I_{1} = I_{1} \cap \{ TC_{i}^{(1)} \}; \]

oD;

\[ \forall \forall j : TC_{i}^{(1)} \in I_{3} : \text{Do} \]
\[ TC_{i}^{(1)} .tc.begin \leftarrow \min (TC_{i}^{(1)} .tc.begin, v.end); \]
\[ TC_{i}^{(1)} .tc.end \leftarrow TC_{i}^{(1)} .tc.end; \]
\[ TC_{i}^{(1)} .tc.begin \leftarrow \text{constraint.tc.begin}; \]
\[ TC_{i}^{(1)} .tc.end \leftarrow \text{constraint.tc.end}; \]
\[ I_{3} = I_{3} \cap \{ TC_{i}^{(1)} \}; \]

oD;

\[ P = \{ TC_{i}^{(1)} : (TC_{i}^{(1)} \in I_{1}) \lor (TC_{i}^{(1)} \in I_{3}) \lor (TC_{i}^{(1)} \in I_{4}) \}; \]

\[ \forall \forall j : TC_{i}^{(1)} \in P : P = \bigcup_{i \neq j} TC_{i}^{(1)} .P; \]

33
/* Check schedulability condition and accept if possible. */
if schedule_type=non_preemptive then Do
    if (result ← TCin.P \subseteq P^*) then
        Already_Accepted ← Already_Accepted \cup \{TCin\};
        return(result);
endif;

if schedule_type=preemptive then Do
    S_y ← S(I^p \subseteq P \cup \{TCin\} \setminus \{TC_s\});
    if (result ← \forall s_e \in S_y:
            (\forall i : TC_i \in I^p : \sum_{k : P_i \in P} ||s_e \cap P_i^{(k)}|| \leq ||s_e||)
            ^ (\sum ||s_e|| \leq \|v\| - \sum_k ||P_i^{(k)}||) then
        Already_Accepted ← Already_Accepted \cup \{TCin\};
    endif;
endif;
endif;

4 Properties

We assume that each of the already accepted time constraints in the calendar have passed successfully 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" \not\subset 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\subset TC' : When checking the feasibility for TC", a portion of TC' \cap 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_s.

   \exists s_e \in S(I^p) : TC' \cap s_e \neq \emptyset \lor TC' = TC_s.

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 Algorithms

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

\[
\text{repeat} \cdots \text{until } I_4 \neq \phi.
\]

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

\[
I_4 = \{TC'_k\}, \quad k = 1, \ldots
\]

such that

\[
\forall k : TC'_k \gg TC_{in}.
\]

We therefore set \( v \) to \( TC'_{\text{max}} \) such that

\[
\exists TC'_k' \in I_4 : TC'_k' \gg TC'_{\text{max}}
\]

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

\[
\exists TC' : TC' \gg TC'_{\text{max}}.
\]

But then \( TC' \) also satisfies

\[
TC' \gg TC_{in}
\]

which implies

\[
TC' \in \{TC'_k\}, \quad k = 1, \ldots
\]

that contradicts the definition of \( TC'_{\text{max}} \).
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 schedulability 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 \( \geq 0 \) we can write

\[
\forall P_i \in P^o : P_i \uparrow TC_i \lor P_i \ll TC_i \lor P_i \downarrow TC_i \lor P_i = TC_i,
\]

and

\[
P_{in} \uparrow TC_{in} \lor P_{in} \ll TC_{in} \lor P_{in} \downarrow TC_{in} \lor P_{in} = TC_{in}.
\]

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

\[
\forall i,j : P_i, P_j \in P^o \land i \neq j : P_i \propto P_j.
\]

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

\[
\exists T_s : P_i\|T_s\|P_{in} \lor P_{in}\|T_s\|P_i^o,
\]

with \( \|T_s\| > 0 \). The existence of such a \( T_s \) for computation \( P_i \), whose corresponding window of occurrence \( TC_i \) satisfies \( TC_i \in I_i^o \), implies the existence of another interval \( T_y \), for computation \( P_i \).

\[
\exists T_y : P_i\|T_y\|P_{in} \lor P_{in}\|T_y\|P_i.
\]

\( P_i \)'s corresponding window of occurrence, \( TC_i \), satisfies \( TC_i \in I_i \). The existence of \( T_y \) is implied, due to the way \( I_i^o \) is constructed from \( I_i \) by shifting \( P_i \) to the right to produce \( P_i^o \), such that

\[
T_s \downarrow T_y \lor T_s = T_y.
\]
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

\[ \exists T_s : P_i \| T_s \| P_i \lor P_i \| T_s \| P_i, \]

for computation interval \( P_i \), whose corresponding window of occurrence \( TC_{i_o} \) satisfies \( TC_{i_o} \in I_3 \). Hence, by the way \( I_5 ^o \) is constructed from \( I_3 \), we can have a \( T_s \)

\[ T_s \uparrow T_s \lor T_s = T_s \]

such that

\[ \exists T_s : P_i \| T_s \| P_i , \lor P_i \| T_s \| P_i, \]

for computation interval \( P_i \), whose corresponding window of occurrence \( TC_i \) satisfies \( TC_i \in I_5 \).

### 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 \( TC_{i_o} \). 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. \( TC_{i_o} = TC_{i_o} \): From the first test in Condition 2 we know that for those \( s_o \) for which \( s_o = \bigcup TC_{i_o} \)

\[ \| s_o \| \geq \sum_{v_i \in \Theta} \| P_i \| \cdot \]

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

\[ \sum_{v_i \in \Theta} \| s_o \| \geq \sum_{v_i \in \Theta} \sum_{v_k \in \Theta} \| P_{i_o} \| \cdot \sum_{v_k \in \Theta} \| P_i \cap TC_{i_o} \| \cdot \]

The right side of the above equation originates in the construction of \( I_3 ^o \) and \( I_5 ^o \). Equality holds for \( I_5 \), as well as for members of \( I_3 ^o \) and \( I_5 ^o \) 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 $TC_{in}$

$$\|TC_{in}\| = \|V\| \geq \sum_{v_{a_{e}} \in S(I_{V})} \|s_{a_{e}}\| + \|P_{in}\| \geq \sum_{v_{a_{e}} \in S(I_{V})} \sum_{W_{i}(s_{a_{e}})} \|P_{i}^{o}\| + \|P_{in}\|$$

and we can conclude by replacing $\sum_{v_{a_{e}} \in S(I_{V})} \sum_{W_{i}(s_{a_{e}})}$ with $\sum_{W_{i}:TC_{i} \in I^{o}}$ to get

$$\|TC_{in}\| \geq \sum_{W_{i}:TC_{i} \in I^{o}} \|P_{i} \cap TC_{in}\| + \|P_{in}\|.$$  

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

2. $TC_{s} \neq TC_{in}$: 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 $\forall s_{a_{e}} \in S(I_{V})$ for those $i$'s for which $s_{a_{e}} = \bigcup_{i(s_{a_{e}})} TC_{i}^{o}$

$$\|s_{a_{e}}\| \geq \sum_{W_{i}(s_{a_{e}})} \|P_{i}^{o}\|.$$  

In this case

$$TC_{i}^{o} \in I^{o} \lor TC_{i}^{o} = TC_{in}.$$  

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

$$\sum_{v_{a_{e}} \in S(I_{V})} \|s_{a_{e}}\| \geq \sum_{v_{a_{e}} \in S(I_{V})} \sum_{W_{i}(s_{a_{e}})} \|P_{i}^{o}\|$$

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

$$\sum_{v_{a_{e}} \in S(I_{V})} \|s_{a_{e}}\| \geq \sum_{v_{a_{e}} \in S(I_{V})} \sum_{W_{i}(s_{a_{e}})} \|P_{i}^{o}\| + \|P_{in}\|.$$  

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

$$\sum_{v_{a_{e}} \in S(I_{V})} \|s_{a_{e}}\| \geq \sum_{v_{a_{e}} \in S(I_{V})} \sum_{W_{i}(s_{a_{e}})} \|P_{i}^{o}\| + \|P_{in}\| \geq \sum_{v_{a_{e}} \in S(I_{V})} \sum_{W_{i}:TC_{i} \in I^{o} \land \left\{TC_{s}\right\}} \|P_{i} \cap TC_{s}\| + \|P_{in}\|.$$  

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

$$\|TC_{s}\| = \|V\| \geq \sum_{v_{a_{e}} \in S(I_{V})} \|s_{a_{e}}\| + \|P_{s}\|.$$  

Using our results from above

$$\|TC_{s}\| = \|V\| \geq \sum_{v_{a_{e}} \in S(I_{V})} \sum_{W_{i}:TC_{i} \in I^{o} \land \left\{TC_{s}\right\}} \|P_{i} \cap TC_{s}\| + \|P_{s}\| + \|P_{in}\|.$$  

38
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.

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

and

$$\bigcup_j P^{(j)} \uparrow TC_{in} \lor \bigcup_j P^{(j)} \ll TC_{in} \lor \bigcup_j P^{(j)} \downarrow TC_{in} \lor \bigcup_j P^{(j)} = TC_{in}.$$ 

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

$$\forall i, j : P_i, P_j \in P^o \land i \neq j : P_i \cap P_j.$$ 

As in the convex case, one can therefore infer the existence of an idle interval $T_s$ between the incoming computation $P_{in}$ and each of the already accepted computations $P_i \in P^o.$

In other words

$$\forall k, n : P^{(k)}_{in} \in P_{in} \land P^{(n)}_{in} \in P_{in} :$$

$$\exists T^{(k,n)}_s : P^{(k)}_{in} \parallel T^{(k,n)}_s \parallel P^{(n)}_{in} \lor P^{(n)}_{in} \parallel T^{(k,n)}_s \parallel P^{(k)}_{in},$$

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

For every $P_i$ that corresponds to $TC_{i_e} \in I_2$ the existence of

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

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

$$\forall k, n : T^{(k,n)}_s \downarrow T^{(k,n)}_y \lor T^{(k,n)}_s = T^{(k,n)}_y.$$
Thus, we can use this property to conclude $\|T_y^{(k,n)}\| \geq \|T_y^{(k,n)}\|$; that yields

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

which assures a feasible schedule. Similarly, for every $TC_i \in I^o$

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

Since $\|T_y^{(k,n)}\| \geq \|T_y^{(k,n)}\|$, 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. $TC_s = TC_{in}$: From the first test in Condition 4 we know that $\forall s_0 \in S(I^o)$ for those $i$'s for which $s_0 = \bigcup_i TC_i^o$

$$\|s_0\| \geq \sum_{\forall i(s_0) \forall k: P_i^{(k)} \in P_i^o} \|P_i^{(k)}\|.$$

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

$$\sum_{\forall s_0 \in S(I^o)} \|s_0\| \geq \sum_{\forall s_0 \in S(I^o)} \sum_{\forall i(s_0) \forall k: P_i^{(k)} \in P_i^o} \|P_i^{(k)}\| \geq \sum_{\forall s_0 \in S(I^o)} \sum_{\forall TC_i \in I^o} \sum_{\forall n: P_i^{(n)} \in P_i^o} \|P_i^{(n)} \cap TC_{in}\|.$$

If the second test in Condition 4 holds, then

$$\|TC_{in}\| = \|V\| \geq \sum_{\forall s_0 \in S(I^o)} \|s_0\| + \|P_{in}\| \geq \sum_{\forall s_0 \in S(I^o)} \sum_{\forall i(s_0) \forall k: P_i^{(k)} \in P_i^o} \|P_i^{(k)}\| + \|P_{in}\|$$

and we can conclude by replacing $\sum_{\forall s_0 \in S(I^o)} \Sigma_i(s_0)$ with $\sum_{\forall TC_i \in I^o}$ to get

$$\|TC_{in}\| \geq \sum_{\forall TC_i \in I^o} \sum_{\forall n: P_i^{(n)} \in P_i^o} \|P_i^{(n)} \cap TC_{in}\| + \|P_{in}\|.$$
2. \( TC_s \neq TC_{in} \): As in the above case, the first test in Condition 4 yields

\[
\| \delta_v \| \geq \sum_{\forall i(e)} \| P_i^{(s)} \|.
\]

Here

\[
TC_i^{(s)} \in I^o \lor TC_i^{(s)} = TC_{in}.
\]

Summing

\[
\sum_{\forall i(e) \in S(I_Y)} \| \delta_v \| \geq \sum_{\forall i(e) \in S(I_Y)} \sum_{\forall k: P_i^{(s)}(e) \in P_{i(e)}} \sum_{\forall k: P_i^{(s)}(e) \in P_{i(e)}} \| P_i^{(s)} \| + \| P_{in} \|.
\]

where one of these \( P_i^{(s)}(e) \) is \( P_{in} \), and the rest are members of \( I^o \). Hence,

\[
\sum_{\forall i(e) \in S(I_Y)} \| \delta_v \| \geq \sum_{\forall i(e) \in S(I_Y)} \sum_{\forall k: P_i^{(s)}(e) \in P_{i(e)}} \sum_{\forall k: P_i^{(s)}(e) \in P_{i(e)}} \| P_i^{(s)} \| + \| P_{in} \|.
\]

As in the convex case, it can be extended to

\[
\sum_{\forall i(e) \in S(I_Y)} \| \delta_v \| \geq \sum_{\forall i(e) \in S(I_Y)} \sum_{\forall k: P_i^{(s)}(e) \in P_{i(e)}} \sum_{\forall k: P_i^{(s)}(e) \in P_{i(e)}} \| P_i^{(s)} \| + \| P_{in} \|.
\]

From the second test in Condition 4

\[
\| TC_s \| = \| V \| \geq \sum_{\forall i(e) \in S(I_Y)} \| \delta_v \| + \sum_{\forall k: P_i^{(s)}(e) \in P_{i(e)}} \| P_i^{(s)} \|.
\]

Using our results from above

\[
\| TC_s \| = \| V \| \geq \sum_{\forall i(e) \in S(I_Y)} \sum_{\forall k: P_i^{(s)}(e) \in P_{i(e)}} \sum_{\forall k: P_i^{(s)}(e) \in P_{i(e)}} \| P_i^{(s)} \| + \| TC_s \|
\]

\[
+ \sum_{\forall k: P_i^{(s)}(e) \in P_{i(e)}} \| P_i^{(s)} \| + \sum_{\forall k: P_i^{(s)}(e) \in P_{i(e)}} \| P_i^{(s)} \|.
\]

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 introduced 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

$$ A = (t_a^a, t_b^A), \quad B = (t_a^b, t_b^B). $$

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

- **equal ($A = B$)**
  $$ t_a^A = t_a^B \wedge t_b^A = t_b^B, $$

- **preceed ($A < B$) and its inverse succeed ($A > B$)**
  $$ t_b^A < t_a^B, $$

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

- **overlap ($A \odot B$) and its inverse overlapped-by ($B \odot^u A$)**
  $$ t_a^A < t_a^B < t_b^A < t_b^B, $$

- **start ($A \Uparrow B$) and its inverse started-by ($B \Uparrow^u A$)**
  $$ t_a^B = t_a^A < t_b^A < t_b^B, $$

- **during ($A \lhd B$) and its inverse contain ($B \rhd A$)**
  $$ t_a^B < t_a^A < t_b^A < t_b^B, $$

- **end ($A \Downarrow B$) and its inverse ended-by ($B \Downarrow^u A$)**
  $$ t_b^A < t_a^A < t_b^B = t_a^B. $$

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

$$ A \bowtie B \equiv (A < B) \vee (A > B), $$

and all the containment possibilities as

$$ (A \Uparrow B) \vee (A \lhd B) \vee (A \Downarrow B). $$
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-$\mathcal{R}^\prime$ $C$, where $\mathcal{R}^\prime$ is the converse relation to $\mathcal{R}$.

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

\[ \forall d_j \in D : \exists c_i \in C : c_i \Rightarrow d_j, \]
and $D$ mostly-$\Leftarrow$ $C$

\[ \forall c_i \in C : \exists d_j \in D : d_j \Leftarrow c_i. \]

- **partially**: $C$ partially-$\mathcal{R}$ $D$ if and only if
  \[ X = \{x_i\} = \{c_i, d_i : c_i \mathcal{R} d_i\} \neq \emptyset, \]
  \[ \{C - X\} \cap \{D - X\} = \emptyset. \]

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
  \[ X = \{x_i\} = \{c_i, d_i : c_i \mathcal{R} d_i\} \neq \emptyset. \]

- **disjunction**: $\mathcal{R} \lor ... \lor \mathcal{Q}$ $D$ if and only if
  \[ \forall c_i \in C, \forall d_j \in D : c_i \mathcal{R} d_j \lor ... \lor c_i \mathcal{Q} d_j \lor \exists c_i \approx d_j. \]

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 ($\nabla$) relation, that can be expressed as "totally-$\nabla$", meaning
\[ \forall d_j \in D : \forall c_i \in C : c_i \nabla d_j. \]

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 $C \bar{\leftrightarrow} D$, whose union is convex.
  
  $C \leftrightarrow D$ if and only if $C \cup D = C \cup D$.

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,

$C$ totally-$\prec D$ 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 $C \bar{\uprightfig} D$, if and only if $\triangleright C \uprightfig \triangleleft D$.

Another example is the containment property. We have generalised above one possible containment with $C \rightarrow D$ (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,

- **surrounded**: non-convex interval $C$ surrounds non-convex interval $D$, denoted $C \bowtie D$, if and only if $\bigcup\{C\} \bowtie \bigcup\{D\}$.

For allowing reasoning on subintervals, we therefore write
\[
((\triangleleft C \prec \triangleleft D) \lor (\triangleleft C \bowtie \triangleleft D) \lor (\triangleleft C \ll \triangleleft D) \lor (\triangleleft C \ll \triangleleft D)) \lor \\
((\triangleright C \gg \triangleright D) \lor (\triangleright C \bowtie \triangleright D) \lor (\triangleright C \ll \triangleright D) \lor (\triangleright C \ll \triangleright D)).
\]

The expression is much simpler for the disjoint case
\[(C \nabla D) \land (\triangleleft C \prec \triangleleft D) \land (\triangleright C \gg \triangleright D).
\]
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:** $C \preceq D$ if
  $$(\prec C \prec D \lor (\prec C \lhd D) \lor (\prec C \lhd D) \lor (\prec C \lhd D) \lor (\prec C \lhd D)).$$

- **start-after:** $C \succ D$ if
  $$(\prec C \succ D \lor (\prec C \rhd D) \lor (\prec C \rhd D) \lor (\prec C \rhd D) \lor (\prec C \rhd D)).$$

- **start-at:** $C \equiv D$ if
  $\prec C \equiv D.$

- **end-at:** $C \equiv D$ if
  $\succ C \equiv D.$

- **end-after:** $C \succ D$ if
  $$(\succ C \succ D \lor (\succ C \rhd D) \lor (\succ C \rhd D) \lor (\succ C \rhd D) \lor (\succ C \rhd D)).$$

- **end-before:** $C \preceq D$ if
  $$(\succ C \preceq D \lor (\succ C \rhd D) \lor (\succ C \rhd D) \lor (\succ C \rhd D) \lor (\succ C \rhd D)).$$

Finally, a very important classification 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:** $C \cap D$ if and only if
  $$(C \cap D) \land (\prec C \prec D) \land (\succ C \succ D) \land (\prec D \prec D) \land (\succ D \succ D).$$

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
$$((C \cap D) \land (C \supset D)) \lor (C \cap D)$$ can be a proper schedulability condition.
References


Temporal Relations and Structures in Real-Time Operating Systems

Shem-Tov Levi and Ashok K. Agrawala

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.
END DATE FILMED
6-1988 DTIC