Issues in Temporal Reasoning for Autonomous Control Systems *

Nicola Muscettola ‡
Paul Morris †
Barney Pell ¶
Ben Smith §

Abstract

Deep Space One will be the first spacecraft to be controlled by an autonomous agent potentially capable of carrying out a complete mission with minimal commanding from Earth. The New Millennium Remote Agent (NMRA) includes a planner-scheduler that produces plans, and an executive that carries them out. In this paper we discuss several issues arising at the interface between planning and execution. including execution latency, plan dispatchability, and the distinction between controllable and uncontrollable events. Temporal information in the plan is represented within the general framework of Simple Temporal Constraint networks, as introduced by Dechter, Meiri, and Pearl. However, the execution requirements have a substantial impact on the topology of the links and the propagation through the network.

1 Introduction

Deep Space One (DS1) will be the first spacecraft to be controlled by an autonomous agent, the New Millennium Remote Agent (NMRA) [15], potentially capable of carrying out a complete mission with minimal commanding from Earth. Similarly to other high-level control architectures [2] [23] [7] [20] [14] NMRA clearly distinguishes between a *deliberative* layer and a *reactive* layer in its architecture. The Planner/Scheduler (PS) [12] is in charge of deliberative problem solving. PS transforms a set of goals and constraints between goals into a complete plan. This activity may require extensive reasoning about goals and task interdependencies, efficient utilization of resources and satisfaction of temporal deadlines. Similarly to classical planning [19] and scheduling [8] systems this may involve extensive search and it is inherently time consuming. Once it is produced, the plan is shipped to the Executive (EXEC) [17, 18, 16], which issues commands to the low-level real-time control software according to the plan's directives. EXEC is purely *reactive*. High-level directives contained in the plan correspond to detailed execution scripts. Such scripts allow EXEC to locally react to "sensor information" interpreted by the Model-Based Identification and Recovery (MIR) system [24]. However, if EXEC or MIR determine that the execution conditions of the plan are not being met, EXEC does not try to rearrange the plan in order to adapt to the new situation. Instead, plan execution is interrupted, the spacecraft is put in a safe, standby state and PS is asked for a new plan reflecting the new spacecraft configuration.

As a consequence of the different type of problem solving of PS and EXEC, the requirement on constraint propagation algorithms is very different. PS is mainly interested in checking the temporal consistency of plans whose topology is modified at every step of the search process. In general, topology modifications are not restricted to any specific area of the plan and they typically move around the plan quite dramatically during search. The temporal representation required by PS must use minimal temporal descriptions and fast global consistency checking. On the other hand, while EXEC works with plans of a fixed topology, EXEC is interested in doing the minimal amount of computation required in order to *keep* consistency of the plan during execution. Consistency maintenance also has very strong locality properties since EXEC always executes activities in a strictly monotonic temporal order (from past to future). Tight requirements on EXEC's reactivity, however, make the propagation algorithms used by PS impractical. In this respect, minimality of the temporal networks of a plan is not a major issue and redundant information may actually be needed in order to improve performance.

In the rest of the paper we describe those aspects of temporal reasoning embodied in NMRA's PS and EXEC that we found to be crucial to ensure performance and a robust execution in an autonomous control system operating in real-time. First we briefly describe the temporal representation and temporal reasoning mechanisms used by PS (Section 2) and EXEC (Section 3). Then we discuss how to make a plan executable by the simple temporal dispatching propagation used by EXEC (Section 4). We then describe the problem of

*This paper describes work partially performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract from the National Aeronautics and Space Administration.

‡Recom Technologies, NASA Ames Research Center, MS 269/2, Moffett Field, CA 94035.

†Caelum Research, NASA Ames Research Center, MS 269/2, Moffett Field, CA 94035.

¶Riacs, NASA Ames Research Center, MS 269/2, Moffett Field, CA 94035.

§Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109.

handling events over which EXEC does not have complete control and we sketch a possible solution (Section 5). We then discuss related work and conclude the paper.

2 Planning and temporal propagation in PS

Deliberative problem solving in NMRA is carried out by the Planner/Scheduler (PS). PS consists of two components. The Incremental Refinement Scheduler (IRS) is an interval based temporal planner consisting of a heuristics-driven sub-goaling search. IRS is implemented on top of HSTS, a domain modeling and temporal database system [10]. Like other temporal planning representations [1] [5] HSTS imposes the restriction that all assertions (called literals and actions in other planning frameworks) are temporally scoped, i.e., are valid over an interval. We call a temporally scoped assertion a *token*.

The domain representation of HSTS differs from that of other planning systems in two ways. First, HSTS imposes the restriction that each token must be associated with a *state variable*. A state variable can be considered as a generalization of a single capacity resource and imposes the restriction that only one token can be associated with a state variable at any given time. The activity of each state variable in a plan is represented by a *timeline*, which consists of a sequence of tokens. Each token has a non-negative duration and tokens may be synchronized by appropriate temporal relations. For example (Figure 1), in a spacecraft domain one plan timeline may describe the state of the engine (warming up, thrusting, or idle) and another the spacecraft attitude (e.g., pointing to a target or turning from target A to target B). Timelines are completely filled by sequences of tokens without intermediate gaps. The transitions between tokens are called *timepoints* and constitute the variables of a simple temporal network.

The second major difference between HSTS and other planning representations is that HSTS does not see the world as an alternation between *actions* and *states*, with actions and states represented by literals in the domain theory. Instead, HSTS sees the world as the continuous evolution of parallel *threads*, one for each state variable. Since plans are indeed programs to be executed by EXEC, the literal associated with a token, the *token type*, is in fact the invocation pattern of a function that EXEC will be running while the token is active. For each thread, execution will be transferred from one function to another at a timepoint. Loosely speaking, a timepoint is therefore the analogue of an "action" in classical planning. It has no temporal extension and represents the moment at which the function corresponding to the new token starts executing.

Besides filling in gaps between tokens on a timeline, PS must also enforce synchronization constraints between tokens that are required by the domain theory. These constraints are represented as *compatibilities* associated with each token type that can possibly belong to a state variable.

Figure 2 shows a simple compatibility tree derived from the DS1 domain theory. It says that the state in which the MICAS camera is on must be preceded by a state in which it is turning on, and followed by one in which it is turning off. While the camera is on, it consumes fifteen watts of power.

Temporal relations requested by a compatibility are implemented by adding a set of temporal distance constraints between the timepoints corresponding to the start and end of the tokens involved. For example, a relation such as

$$X \ \mathbf{contained\_by} \ [\mathbf{10}, \mathbf{30}] \ [\mathbf{0}, +\infty] \ Y$$

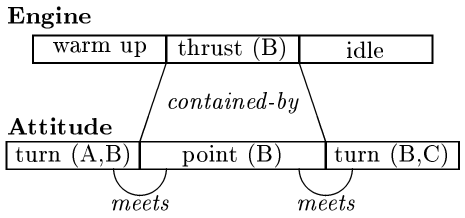

Figure 1: Plans as Parallel Timelines.

```
(MICAS_On)
  :compatibilities
  (AND
    (met_by (MICAS_Turning_On))
    (meets  (MICAS_Turning_Off))
    (equal  (REQUEST (Power 15)))
```

Figure 2: A Compatibility Tree.

will be implemented as

$$10 \leq \mathbf{start}(X) - \mathbf{start}(Y) \leq 30$$

and

$$0 \leq \mathbf{end}(Y) - \mathbf{end}(X) \leq +\infty$$

. The HSTS domain description language allows the representation of every possible combination of distance constraints between the timepoints in one token and the timepoint in another token. Other separation links between timepoints are also imposed between the start and the end of a single token. These represent temporal constraints on the *duration* of a token and are required to be non-negative. The temporal relations implemented between two tokens are guaranteed to be convex; therefore, disequalities between timepoints are not allowed. The set of timepoints and distance links thus constitute a simple temporal network. It is well known [6] that the solution of such a constraint network can be translated into the solution of a set of associated shortest-path problems.

During plan construction PS repeatedly extends and modifies a partial plan. These modifications induce changes to the temporal constraint network (timepoints, distance links and distance bounds). In order to guarantee plan consistency the plan needs to propagate frequently to ensure that the addition or removal of constraints and variables has not created an inconsistency situation. For this purpose, HSTS only uses temporal propagation that is equivalent to an incremental version of the Bellman-Ford algorithm [4]. To ensure high performance, the propagation algorithm includes fast inconsistency detection and influence analysis to isolate minimal required modifications during propagation, along the lines of [3]. Another optimization used is to always represent sets of equivalent timepoints with a single timepoint variable. Collapse and separation of timepoints are performed every time an equality constraint (i.e., a separation link with bound $[0, 0]$) is added or removed from the constraint network.

3 Execution and temporal propagation in EXEC

Given a simple temporal network, EXEC has two important duties:

1. Select and schedule activities for execution.
2. Update the network to reflect new data.

These two duties are the core of the "plan running" algorithm. Basically, the plan runner manages a queue of timepoints that become available for execution as time passes and logically preceding timepoints are executed. The actual execution time of a timepoint is used to propagate temporal information in the plan. (This is one of the sources of "new data" to which item 2 refers.) Together with the simple passage of time, timepoint execution directly narrows the domain of the executed timepoint while other timepoint domains may be indirectly affected because of the network constraints. To avoid future inconsistency, the changes must be propagated through the network.

While EXEC is performing this cycle, other timepoints in the plan can become executable simply because time has passed. These will appear in the queue of incoming timepoints but EXEC may not get to them until the plan running cycle is completed. This cycle constitutes an intrinsic EXEC *latency* which is in fact a quantitative measure of EXEC's reactivity. It is important for PS to know EXEC's latency for two reasons. First, latency is equivalent to execution time *uncertainty.* If a timepoint is scheduled to start at time t_0 and the latency is λ, EXEC can only guarantee that the timepoint will indeed be executed at time t with $t_0 - \frac{\lambda}{2} \leq t \leq t_0 + \frac{\lambda}{2}$. Second, the latency determines the possible lengths of temporal links that can separate two time points in the plan in order to guarantee a robust plan execution. This imposes a restriction on the temporal constraints that can be included in the PS model or that can be requested by a user. Basically, such temporal constraints can only include zero temporal separations (handled by PS as timepoint equivalences) and temporal separations larger than the latency. If the maximum requested separation between two timepoints t_1 and t_2 is smaller than the latency, then EXEC cannot guarantee that the plan will execute without breaking. In fact, by the time t_2 is enabled the separation between the current time and scheduled time for t_2 could be greater than the latency, a situation that will break plan execution.

Notice that the concept of *latency* or *time uncertainty* is quite different from that of *time granularity.* The latter is the minimum separation between time events that can be recognized by the *time services clock*, the device that provides time both to NMRA and to the underlying real-time control system. PS can indeed schedule timepoints separated by distances shorter than the time uncertainty *provided* that these time points are not connected by an explicit enablement link. The reason for this apparently counterintuitive behavior is that the existence of a link guarantees that the following timepoint will have to be in scope in the next EXEC cycle, while unconnected timepoints can be correctly handled in the same cycle. Clearly, the time uncertainty for the execution of such events is still equal to the EXEC latency.

In NMRA, EXEC latency is conservatively estimated to be at least the expected worst case performance under the most taxing conditions (corresponding to starting a token for each of the plan timelines). More adaptive schemes are also possible in which PS schedules very short duration tokens only if the number of timepoints that must be started

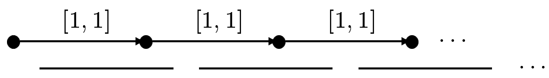

Figure 3: Accumulation of Latency.

at the same time is low according to the current partial plan and therefore EXEC's latency is lower.

For EXEC to be highly reactive, latency should be as low as possible. Because propagation time has a major impact on the EXEC latency, it is essential that propagation time be minimized. We achieve this by restricting the EXEC to one-step propagation, i.e., the propagation is only carried as far as the immediate neighbors. EXEC's time propagation proceeds as follows. EXEC receives from the planner a directed acyclic graph of timepoints, called the *activity graph*, which has the form of a simple temporal network. We will say a timepoint is *scheduled* once its actual time of occurrence becomes fixed and known. A timepoint in the graph is *enabled* if all its parents have already been scheduled. Once it is enabled, a timepoint becomes itself eligible for scheduling. After a timepoint is scheduled, the EXEC must propagate the "collapsed" time-value to other timepoints. For enabled timepoints that are not scheduled, the EXEC should also propagate the narrowing of the lower time bound that corresponds to the passage of time. While this design simplifies propagation and hence reduces latency, it requires plans that execute correctly in spite of this restriction, as will be discussed in the next section.

One final point relates to the relation between t_e, the actual time at which a timepoint is executed, and t_s, the scheduled time. The latter corresponds to the time at which EXEC actually *declares* that the timepoint has been executed. This time is used for time propagation in the plan and, because of EXEC's latency, it is *different* than the actual execution time. Specifically, consider a time point with time bound $[l, U]$ and assume that λ is the EXEC latency. Then $t_s = \max(t_e - \frac{\lambda}{2}, l)$. In other words the scheduled time is pushed in the past by at most half the time uncertainty. Without this distinction, timepoint execution will be subject to the problem of *latency drift.* Latency will accumulate in the plan during EXEC's time propagation and the propagated time bounds of timepoints late in the plan will be pushed (and eventually collapsed) into the upper bound U. The translation of execution time into the past is consistent with the intrinsic time execution uncertainty and avoids the latency drift problem.

The example in figure 3 illustrates what can happen if the actual time of execution is propagated. Here there is a tightly synchronized sequence of timepoints. Although the sequence as a whole may have a flexible time window, the latencies accumulate and eventually overwhelm the flexibility.

4 Dispatchability

The activity graph is essentially the final form of the simple temporal network used by the planner in constructing the plan. However, certain restrictions and modifications are necessary. Most importantly, because of one-step propagation, some additional links may be needed to detect potential inconsistencies before it is too late. As an example of this consider figure 4.

Once A has been scheduled, both D and B are enabled.

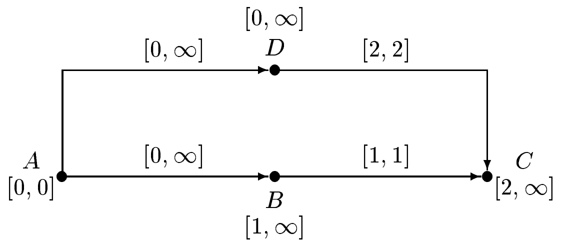

Figure 4: B forces D into past.

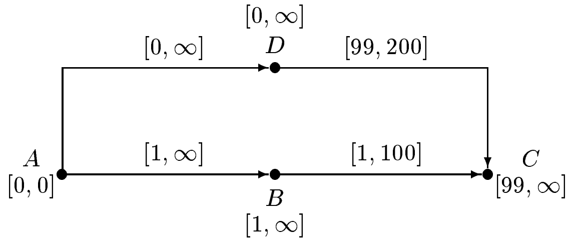

Figure 5: Prefer B after D.

Thus, B may be scheduled before D. Here it is easy to see that D must occur 1 time unit before B. Unfortunately, this may not be discovered until some time after B has been scheduled, when the collapsed value has been propagated to D.

It can be shown that if one timepoint is forced to be after another, then its earliest-start-time (EST) must be later. Thus, one solution to this difficulty is to schedule activities at their earliest start times. However, that would severely restrict the flexibility of the plan since all activities would have their time bounds collapsed to a single time. The solution we adopt is to place an additional "implied" link from D to B. In that case, if B is scheduled before D, the inconsistency will be discovered immediately. In general, the additional links are only used for propagation, not for determining enabling conditions. However, in cases where the minimum separation is non-negative, as here, the new link could also be used as an enabling condition.

It is shown in [13] that the network, augmented with implied links, can be filtered to a smaller size by removing unneeded edges; in practice, the filtered size turns out to be comparable to that of the original network.

At the cost of some extra work for the EXEC, the implied links could also be used to select between several enabled activities. Consider the network in figure 5. In this case, the distance for DB computes as $[-1, 199]$, which means D and B can occur in any order. However, if B occurs first then D must follow within 1 time unit, whereas if D occurs first then B may occur at any time within 199 time units. Clearly the second option retains greater flexibility, which suggests using the distance information to help guide the execution.

5 Uncontrollable Events

We now consider *uncontrollable* timepoints whose time of occurrence is outside the direct control of the EXEC. These may be *observable* or not. The time of observation may follow, or even precede (via predictive measurements), the time of occurrence. The uncontrollable timepoints may be linked either to controllable timepoints or to other uncontrollables. These links may either represent causal relations that are guaranteed by Nature, or constraints that must be respected by the EXEC. In the latter case, the onus is on the EXEC to schedule one end of the link so that the constraint is satisfied.

A general approach for handling uncontrollables is to pretend that EXEC is scheduling natural events, but have it choose times that are consistent with observation. In order for this to work, the scheduling of controllables must not propagate in a way that restricts the flexibility of Nature. In general, a network is considered *brittle* if propagation across a non-causal constraint causes an uncontrollable to be squeezed. In this case, success of the plan represented by the network would depend on a fortuitous occurrence of the uncontrollable.

Unfortunately, it is not enough to ensure that propagation during planning does not squeeze uncontrollables. This is because during execution, timepoints are further squeezed. This happens in three ways.

1. Execution squeezes controllables.
2. Observation squeezes uncontrollables.
3. Simple passage of time may squeeze both.

Thus, the squeeze of a controllable may propagate (via non-causal links) to an uncontrollable during execution, rendering the plan brittle.

The following examples illustrate these points.

- Camera needs to be warmed-up between one and two hours *before* event uncertain by one day.
- Camera needs to be cooled-down between one and two hours *after* event uncertain by one day.
- Camera needs to be warmed-up between one and two *days* before event uncertain by one *hour*.
- Camera needs to be warmed-up when power available before event uncertain.

Note that the first two examples are very similar except for the direction of time. Both produce consistent temporal networks in the sense of Dechter *et al.* At first sight, the first constraint seems impossible to achieve, whereas the second appears easy, because of the direction of time. This assumes the event is observable when it occurs. If we vary this assumption, the symmetry in the cases becomes more apparent. For example, if the time of the event is not observable at all, then the second constraint is equally unachievable. Conversely, in the case of the first constraint, a precursor event may resolve the uncertainty before the warm-up period, so it may be achievable after all. Thus, it is really the information increment resulting from observation that is asymmetric with respect to time. Most commonly, uncontrollable events are regarded as observable at the time they occur.

The third example is clearly achievable even without a precursor event, simply by doing a worst-case analysis.

The last example shows that interactions between requirements can affect the achievability. Depending on other demands for power, this example may resemble either the first or the third.

Aspects of these examples may combine in one problem, as we see in figure 6. Here X is uncontrollable but observable at the time it occurs. We may think of AX as a causal

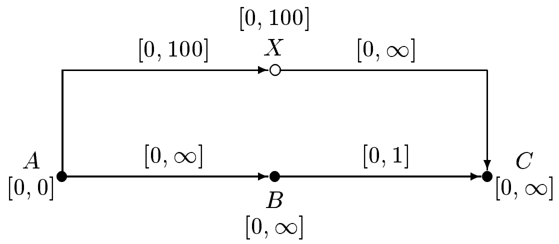

Figure 6: Managing An Uncontrollable.

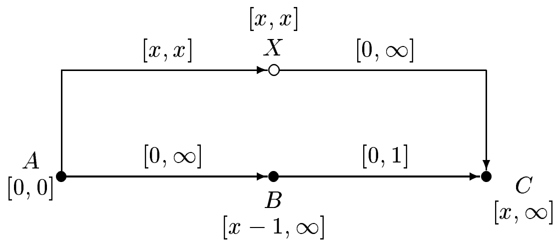

Figure 8: Variable Propagation.

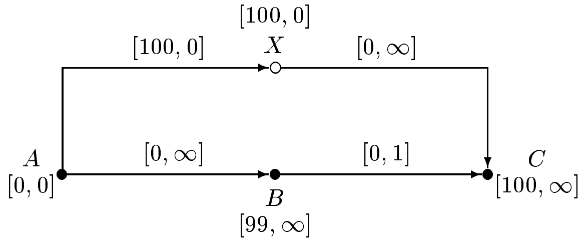

Figure 7: Strong Propagation.

link, while the others are constraints to be achieved. This network is consistent in the sense of Dechter *et al.* Moreover, X is not squeezed by a controllable during the consistency propagation. Note, however, that C is constrained to follow X, while B may not precede C by more than one time unit. Thus, if B is done early, and X happens late, the BC constraint will be violated. To put it another way, the uncontrollable X gets squeezed when B is scheduled.

There are two practical strategies for executing this network. One is to wait until X occurs, and then schedule B. The second is to schedule B at time 99 or later, irrespective of when X occurs. (Even better is a combination of these, where one schedules B either after X has occurred, or on or after time 99.)

Vidal and Fargier [21] consider problems of this kind, and introduce concepts of *strong* and *dynamic* controllability. Roughly speaking, strong controllability means there is a strategy that works irrespective of observation, as in the second case of the previous paragraph. Dynamic controllability may schedule controllables based on observation of uncontrollables, as in the first case. In both cases, the scheduling of a controllable will not squeeze an uncontrollable.

Vidal and Fargier sketch a means for computing a strong controllability solution. The essential idea is to propagate the "non-squeezability" to the controllable timepoints. Thus, the controllables are "super-squeezed" in order to guarantee that the uncontrollables will not be squeezed during execution. This is done by inverting the upper and lower bounds of the duration interval prior to propagation. Thus, if the interval is $[u, v]$, it is propagated as if it was $[v, u]$. Figure 7 shows how this strong propagation would look, as applied to our earlier example. Note that the strong propagation causes X to have $[100, 0]$ as its domain. Note also that backward and forward propagation through the inverted bounds could continue to narrow the domains of A and X indefinitely. It is unclear from Vidal and Fargier's sketchy description how this apparent inconsistency would be excused. In an earlier paper [22], Vidal and Ghallab eliminate the inverted link by, in essence, composing it with a neighbouring controllable link. This restores consistency. (For example, in figure 7, the edges AX and XC can be composed to form the single edge AC with bounds $[100, \infty]$, resulting in a consistent network.) However, they restrict their approach to situations where the uncontrollable links are isolated. A more general treatment that handles arbitrary constellations of uncontrollables would be desirable.

Another method that works for isolated inverted links is to simply exclude *U-turns* in the propagation (i.e., a propagation from Y to X following one from X to Y would be disallowed). With this method (or the composition approach discussed above), strong propagation causes B to have a bound of $[99, \infty]$, which corresponds to one of the execution strategies mentioned earlier.

Vidal and Fargier do not supply a method for deriving a dynamic control strategy. We suggest one here. The basic idea is to assign a variable to represent each uncontrollable duration. The variables may be propagated as algebraic expressions to derive observation-dependent bounds on the controllables. Figure 8 illustrates this. The label $[x-1, \infty]$ of B may be interpreted as saying that B can be scheduled at any time after $x-1$, *as soon as* the value of x is known. If x is observable only when X happens, then the value of $x-1$ is not known until after time x, so we must narrow the domain of B to $[x, \infty]$. This may uncover a "dynamic inconsistency." (Notice, though, that the $x-1$ value potentially handles situations where the value of x becomes known before X actually occurs, such as when a precursor observation provides this information.)

Determining quiescence may appear to pose a difficulty for the task of propagating algebraic expressions. However, one may first compute implied links from each observable node to every controllable node, and then limit the propagation to single steps across those links. The complexity in that case is within the bounds of Johnson's algorithm [4], which appears manageable.

The strong and dynamic strategies can be combined by assigning X a domain label of $[min(x, 100), max(x, 0)]$. In this case, propagation produces $[min(x-1, 99), \infty]$ for B, which may be narrowed to $[min(x, 99), \infty]$, corresponding to the mixed strategy mentioned earlier. Further work is needed to determine if this approach leads to a *test* for dynamic controllability in the sense of Vidal and Fargier, and the resulting complexity. (The problem is conjectured [21] to be NP-complete.)

Strong/variable propagation are important to the planner to produce non-brittle plans. However, the EXEC does not have to perform these costly propagations. The EXEC need only ensure that the narrowing that occurs when an enabled controllable activity is scheduled (or passed over) does not causing squeezing of any uncontrollable. The addition of implied links, as discussed earlier, can ensure that one-step propagation is adequate for this determination. Strategies such as that described in the previous paragraph may also be compiled into restrictions on the EXEC's scheduling of individual activities.

6 Related Work

Temporal reasoning is a fertile area of research in AI. Much of this work centers around isolated reasoning capabilities, such as temporal constraint satisfaction [6], temporal description logics [1], and planning and scheduling with metric time [11]. Our work draws upon these basic capabilities for temporal reasoning, and also addresses additional issues such as execution latency, dispatchability, and uncontrollable events.

Other papers that also deal with the effect of self-time, but in a different setting, include research on negotiating under time constraints [9], *anytime algorithms* [5] to enable the use of results of computation in a timely manner, and the need to reason explicitly about the time taken to produce a plan so that future planning activity can be fit into the current plan [17].

CIRCA [14] is closely related to the work in this paper in that it addresses the need to reason about execution-time computational overhead within the planning process to produce plans that guarantee robust real-time performance. The CIRCA work also addresses the issue of uncontrollable events. A major difference between CIRCA and the current work is that the output of CIRCA is a complex program, whereas the output of our planner is directly executed as a temporal network.

7 Conclusions

This paper has discussed challenging issues that arise when autonomous agents must reason about temporal aspects of the world they inhabit. An agent may need to reason about the passage of time, changes in the world over time, and the impact of such changes on the knowledge and future activities of the agent, and vice-versa. In reasoning about the passage of time, it is especially important that the agent take into account the time required by its own computational process.

We have incorporated these issues into the temporal constraint network paradigm [6]. There are several interrelated conclusions. The latency issue affects the form of updates that are propagated through the network, and reinforces the need for an efficient executive component. This motivates the use of limited propagation, which in turn introduces the need to augment the network with implied links. (An algorithm that minimizes the set of added links is discussed in [13].) The implied links are also useful for supporting stronger forms of propagation that are required to handle scheduling with uncontrollable events.

Acknowledgement We are grateful to Othar Hansson for suggesting examples that helped to stimulate part of this work.

References

[1] J.F. Allen and J.A. Koomen. Planning using a temporal world model. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 741–747, Menlo Park, California, 1983. AAAI Press.

[2] R. P. Bonasso, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. *JETAI*, 9(1), 1997.

[3] A. Cesta and A. Oddi. Gaining efficiency and flexibility in the simple temporal problem. In L. Chittaro, S. Goodwin, H. Hamilton, and Montanari A., editors, *Proceedings of the Third International Workshop on Temporal Representation and Reasoning (TIME-96)*, Los Alamitos, California, 1996. IEEE Computer Society Press.

[4] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT press, Cambridge, MA, 1990.

[5] T. Dean, R.J. Firby, and D. Miller. Hierarchical planning involving deadlines, travel time, and resources. *Computational Intelligence*, 4:381–398, 1988.

[6] R. Dechter, I Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, May 1991.

[7] Brian Drabble, Austin Tate, and Jeff Dalton. O-plan project evaluation experiments and results. Oplan Technical Report ARPA-RL/O-Plan/TR/23 Version 1, AIAI, July 1996.

[8] M.S. Fox and M. Zweben. *Intelligent Scheduling*. Morgan Kaufmann, San Mateo, California, 1994.

[9] Sarit Krause, J. Wilkenfeld, and G. Zlotkin. Multiagent negotiation under time constraints. Technical Report Techreport CS-TR-2975, Computer Science Department, U. of Maryland, October 1992.

[10] N. Muscettola. *HSTS: Integrating Planning and Scheduling*. Morgan Kaufmann, San Mateo, California, 1994.

[11] N. Muscettola. HSTS: Integrating planning and scheduling. In Mark Fox and Monte Zweben, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.

[12] N. Muscettola, B. Smith, S. Chien, C. Fry, G. Rabideau, K. Rajan, and D. Yan. On-board planning for autonomous spacecraft. In *Proceedings of the Fourth International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS)*, July 1997.

[13] Nicola Muscettola, Paul Morris, and Ioannis Tsamardinos. Reformulating temporal plans for efficient execution. In *Proc. of Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, 1998. To Appear.

[14] David Musliner, Ed Durfee, and Kang Shin. Circa: A cooperative, intelligent, real-time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6), 1993.

[15] Barney Pell, Douglas E. Bernard, Steve A. Chien, Erann Gat, Nicola Muscettola, P. Pandurang Nayak, Michael D. Wagner, and Brian C. Williams. An autonomous spacecraft agent prototype. In W. Lewis Johnson, editor, *Proceedings of the First Int'l Conference on Autonomous Agents*, pages 253–261. ACM Press, 1997.

[16] Barney Pell, Ed Gamble, Erann Gat, Ron Keesing, Jim Kurien, Bill Millar, P. Pandurang Nayak, Christian Plaunt, and Brian Williams. A hybrid procedural/deductive executive for autonomous spacecraft. In M. Wooldridge, editor, *Proceedings of the Second Int'l Conference on Autonomous Agents*. ACM Press, 1998. To appear.

[17] Barney Pell, Erann Gat, Ron Keesing, Nicola Muscettola, and Ben Smith. Robust periodic planning and execution for autonomous spacecraft. In *Procs. of IJCAI-97*, Los Altos, CA, 1997. IJCAI.

[18] Barney Pell, Gregory A. Dorais Christian Plaunt, and Richard Washington. The remote agent executive: Capabilities to support integrated robotic agents. In Alan Schultz and David Kortenkamp, editors, *Procs. of the AAAI Spring Symp. on Integrated Robotic Architectures*, Palo Alto, CA, 1998. AAAI Press.

[19] J. S. Penberty and D. Weld. Ucpop: A sound, complete, partial order planner for adl. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–115, San Mateo, California, 1992. Morgan Kaufmann.

[20] Reid Simmons. An architecture for coordinating planning, sensing, and action. In *Procs. DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 292–297, San Mateo, CA, 1990. DARPA, Morgan Kaufmann.

[21] T. Vidal and H. Fargier. Contingent durations in temporal csps: From consistency to controllabilities. *Linköping Electronic Articles in Computer and Information Science*, 2(2), 1997. Available from http://www.ep.liu.se/ea/cis/1997/002/.

[22] T. Vidal and M. Ghallab. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proc. of 12th European Conference on Artificial Intelligence (ECAI-96)*, pages 48–52, 1996.

[23] D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley. Planning and reacting in uncertain and dynamic environments. *JETAI*, 7(1):197–227, 1995.

[24] Brian C. Williams and P. Pandurang Nayak. A model-based approach to reactive self-configuring systems. In *Procs. of AAAI-96*, pages 971–978, Cambridge, Mass., 1996. AAAI, AAAI Press.