TIME MANAGEMENT SITUATION ASSESSMENT (TMSA)*

Michael B. Richardson
miker@aldrin.ksc.nasa.gov
and
Mark J. Ricci
mricci@aldrin.ksc.nasa.gov
Advanced Computing Technologies Group
Boeing Aerospace Operations, FA-71
KSC FL 32899

ABSTRACT
TMSA is a concept prototype developed to support NASA Test Directors (NTDs) in schedule execution monitoring during the later stages of a Shuttle countdown. The program detects qualitative and quantitative constraint violations in near real-time. The next version will support incremental rescheduling, and reason over a substantially larger number of scheduled events.

INTRODUCTION
The Time Management Situation Assessment (TMSA) program is a prototype developed to assist NASA Test Directors (NTDs) manage the later stages of a Shuttle countdown. The NTDs are primarily concerned with the orderly and timely execution of the countdown process. The cognitive model they reason with is a relatively high-level one which includes a nominal (planned) model of the countdown and a set of qualitative and quantitative constraints that define such a countdown by specifying temporal duration and ordinal relationships between countdown events. Constraints vary both in their specificity (e.g. < is more explicit, <= is less explicit) and in their necessity (i.e. from critical - more necessary to desirable - less necessary).

From the perspective of knowledge engineering for TMSA, what is not included in the NTDs' view is as important as what is included. The details of a subsystem or procedural failure, and what is required to correct or bypass it are not, for the purposes of TMSA, a part of the NTDs' view of the countdown situation. Even in an anomalous situation the NTDs' focus remains on the temporal duration and ordinal unfolding of the countdown. When an anomaly occurs the NTDs participate in the anomaly response, primarily, for the purpose of determining the impact the anomaly will have on the temporal and ordinal aspects of the countdown.

The NTDs monitor the current countdown and assess its compliance with their nominal countdown model. When there is a need for a deviation, they consider alternative revisions of the current countdown and assess the legality and desirability of the revised countdown with regard to the constraints. The countdown schedule may be revised by reordering events and/or adjusting the durations of intervals between events.

The existing prototype monitors launch processing during the later stages of the countdown. It detects deviations from a nominal countdown by detecting temporal and prerequisite constraint violations. It then identifies the violated constraint(s). The system is initialized and operates with both qualitative and quantitative constraints on the order of events and intervals, and the duration of intervals.

The prototype is implemented in Smalltalk and runs on a 25mhz 486, under MS DOS. It appears that a C++ version of the program will be able to handle a schedule containing 200-300 events with response times of < 1.5 seconds for each assimilation input (i.e. relation vector refinement).

SALIENT CHARACTERISTICS OF THE SITUATION
In formulating our approach to this scheduling task we found the following characteristics of the situation to be especially important.

*This work is a portion of the technical support provided to the Artificial Intelligence Section, Design Engineering Directorate, by Boeing Aerospace Operations under the Engineering Support Contract at Kennedy Space Center. Arthur E. Beller is the NASA Technical Contact.

1. The situation is highly structured. A pre-existing nominal schedule is available. There is a well formulated, proven set of constraints on the schedule. The horizon for rescheduling is limited by fixed synchronization points which divide and encapsulate the countdown schedule. All possible events in the countdown are known and are of limited number.

2. Although this is an advisory system used by experts, the criticality of the situation places a premium on timeliness and correctness beyond that of many applications. Near real-time (< 1.5 second) responses and an assurance of correctness are required. Rescheduling with verification must be supported with response times, again, in near real-time. The amount of time available for considering schedule alternatives is severely limited, especially near the end of the countdown.

The verification and validation issues in our software environment, along with the above mentioned characteristics led us to approach the problem algorithmically, and avoid using heuristics.

While the countdown is formulated in terms of both events and intervals, the constraints between intervals are such that we have been able to represent intervals as start and end pairs of events. This has permitted us to restrict our representation to a point algebra that along with our variation of the Waltz algorithm provides a reasoning mechanism that is both sound and complete.

KEY CONCEPTS AND DEFINITIONS

Time

From the NTD's perspective countdown time is discrete, with a relatively coarse granularity (i.e. the smallest increments are about one second). Accordingly, we assume a discrete time model and interpret points in time as single integer, and intervals as pairs of integers, with consecutive integers forming the smallest nontrivial intervals. Effectively then, our points are "moments" in the sense of (Allen and Hayes, 1985). A different approach to discrete time and "moments" is described in (Schmiedel, 1990).

Pseudo Events

For several purposes TMSA employs events that are not members of the universe of countdown events employed by the NTDs. As with countdown events, pseudo events have integer time stamps and generally can be manipulated in the same ways as countdown events. Current uses of pseudo events are described below in the Uncertainty discussion.

Uncertainty

Uncertainty arises in the countdown schedule situation in several distinct ways. First of all many of the qualitative constraints between countdown events are ambiguous (e.g. <=). Secondly, ambiguity also occurs in some quantitative duration constraints on the length of intervals.

We represent and reason about quantitative constraints and uncertainty with the same mechanisms used for qualitative constraints and uncertainty. For example, to represent that an event Ej must occur at or after some point in time we generate a pseudo event Ei, time stamp Ei with the appropriate time and establish a constraint relation Rij of <=. This approach extends to duration constraints by using two pseudo events, one for the start and one for the end. By representing quantitative constraints in this way we are able to take advantage of the soundness and completeness of the ConstraintChecker algorithm.

In addition to the nominal countdown model and constraints, the NTDs also employ a quantitative concept of slack time, not unlike that used in project planning systems such as PERT or CPM. For the NTDs slack time is a valuable resource that they seek to preserve for use later in the countdown should it be needed. Currently we do not explicitly represent or reason about slack time, but, we are now examining approaches to representing slack time and evaluating the quality of schedule alternatives in light of the relative preservation of slack each provides.

Finally, there is the usual uncertainty related to confidence in estimates of temporal duration. Currently we do not deal with confidence factors, but, may in the future, when we begin evaluating the quality of schedule alternatives seek some measure theoretic approach to confidence.

Event (Ei):

A primitive object without discrete time duration. Events are used to define the two fundamental types of countdown objects, Intervals and Milestones, and to uniquely represent specific points in discrete time.

Universe of Events:
All the possible events that can occur as part of a countdown. These events are specified in advance to TMSA or are generated pseudo events, and are to be reasoned about by TMSA.

Interval (Iij):
A countdown object with temporal duration (trivially one) defined by two Events Ei and Ej such that if the time stamp associated with Ei is <= Ej then Ei is the start of the Interval Iij and Ej is the finish.

Assertions:
Assertions about Events may be of two types: point assertions about a single Event (e.g. Eventi occurred at time t); and Relationship Assertions about pairs of events (e.g. Eventi <> Eventj).

Quantitative Relation:
A temporal duration between two Events that is expressed as a natural number corresponding to some number of units of discrete time.

Qualitative Relation:
One of the following relationships between two Events: =, <, <=, <>, <=> (unconstrained), () (null). The program converts > to < and >= to <=.

ALGORITHMS
Two algorithms have been developed for TMSA. These form the reasoning Kernel of the program and are designed to monitor and interpret the legality of the temporal duration and sequential unfolding of a countdown.

The first algorithm, ConstraintChecker, is used to maintain a qualitative representation of the current status of a countdown and to check the consistency of that status with the qualitative constraints that define the legality of a countdown.

A popular approach in the scheduling literature is Allen's Interval Algebra (Allen, 1983) and his adaptation of the widely used Waltz Algorithm (Davis, 1987). The ConstraintChecker Algorithm is also an adaptation of the Waltz Algorithm and employs the Point Temporal Algebra presented in (Vilain and Kautz, 1986).

The ConstraintChecker Algorithm deals only with qualitative Relationship Assertions (in the form of Relation Vectors). One of the tasks of the ScheduleMaintainer Algorithm is to generate Relationship Assertions from Point Assertions received from the live data stream or the NTDs.

The second algorithm, ScheduleMaintainer, is used to maintain both a qualitative and quantitative representation of a countdown. the representation includes both the current status of the countdown and the quantitative constraints that define the legality of a countdown. This representation is also used to generate relational assertion vectors as input to the consistency checking algorithm.

ConstraintChecker

ConstraintChecker differs from the Waltz algorithm presented in (Vilain and Kautz, 1986) in two ways. Our algorithm uses an upper diagonal array rather than a n x n array. For our problem we needed to maintain not only a current representation of the constraints/relations between events, but, also the original constraints used to define a nominal countdown. This permits the algorithm to recognize the situation where a change in the relation between two events violates the current relation, but, not the original one. An alternative approach would have been to not update the relations vectors, but only check for validity of the new assertion. We opted for the approach used in order to permit not only the checking of new assertions with the original constraints, but, also to permit the tracking of relation vector changes over time. This capability is useful for debugging the constraint database.

We state the following theorems without the proofs because of space limitations.

The time complexity of ConstraintChecker is $O((n^3)/2)$.
The Space Complexity of ConstraintChecker is $O(n^2)$.
The inference mechanism for ConstraintChecker is sound.
The inference mechanism for ConstraintChecker is complete.

ConArray (constraint array)
An upper diagonal array indexed by events, and in which ConArray[i, j] holds the asserted constraint relationship between events i and j. ConArray holds the defining qualitative constraints (given or generated) that the NTDs use to define a legal countdown. Note that

unlike EmpArray, ConArray is not updated. Thus ConArray maintains a record of the original constraint matrix.

EmpArray (empirical array)
An upper diagonal array indexed by events, and in which EmpArray[i, j] holds the asserted empirical relationship between events i and j. EmpArray holds the current, but, changing relationships (given or generated) that actually occur during the countdown.

EPQueue (event-pair queue)
A FIFO data structure used to keep track of those Pairs of Events for which a changed relationship is asserted.

The addition operation (+) computes the sum of two vectors by finding the common constituent simple relations. This is a means to identify the least restrictive relationship the two vectors together admit. Addition is implemented as a Table lookup and is the same as that presented in (Vilain and Kautz, 1986).

The multiplication operation (x) is defined between pairs of vectors that relate three Events. For example: if Rij relates Events i and j, and Rjk relates Events j and k, the product of Rij and Rjk is the least restrictive relation between i and k that the two vectors together admit. Multiplication is also implemented as a table lookup and is similar to that presented by (Vilain and Kautz, 1986). The table has been reorganized to yield valid results using the upper diagonal array only.

ConstraintChecker

```
Assert (Rij)
/* Rij is a relation being asserted between Ei and
Ej. */
{       Tempij:= EmpArray[ij];
        EmpArray[ij]:= EmpArray[ij] + Rij;
        If EmpArray[ij] ~= Tempij
                Then Put EiEj on EPQueue;     }

Assimilate
/* Monitors EPQueue for new Relationship
Assertions */
{       While EPQueue is not empty Do
                Get next EiEj from EPQueue;
                Propagate (EmpArray[ij]);     }

Propagate (EmpArray[ij])
/* Props new Relation Assertion between Ei and
Ej to other Events */
{       For each Event Ek Do
                Tempij:= EmpArray[ik] +
                (EmpArray[ij] x EmpArray[jk]);
                If Tempij = ()
                        Then ( Check
                        (ConArray[ij]) );
                If EmpArray[ik] ~= Tempij
                        Then Put EiEk on
                        EPQueue;
                EmpArray[ik]:= Tempij;
                Tempj:= EmpArray[jk] +
                (EmpArray[ik] x EmpArray[ij]);
                If Tempij = ()
                        Then ( Check
                        (ConArray[kj];
                If EmpArray[jk] ~= Tempij
                        Then Put EjEk on
                        EPQueue;
                EmpArray[jk]:= Tempij;        }

Check (ConArray[ij])
/* Checks to see if new Relation Assertion
between Ei and Ej, Rij, violates the original
constraint between them*/
{       Tempij:= ConArray[ij];
        ConArray[ij]:= ConArray[ij] + Rij;
        If ConArray[ij] = ()
                Then (signal illegal count);
        If ConArray[ij] ~= Tempij
                Then Replace EmpArray[ij] with
                ConArray[ij] and Put EiEj on
                EPQueue;      }
```

ScheduleMaintainer

ScheduleMaintainer generates qualitative relational assertion vectors by moving an Event data point and time stamp received from an external source into the appropriate position on the multi-linked list that is the central data structure for ScheduleMaintainer. A relational assertion vector (Rij) is generated by taking the moved Event and its new successor as an Event pair EiEj. Quantitative constraints are maintained by using pointers between related Events, Ei and Ej for example, and when Ei is moved, Ej is moved appropriately, and Eventj is then processed as a moved Event, just as the original moved Eventi was processed.

We state the following theorems without the proofs because of space limitations.

The Time Complexity of ScheduleMaintainer is O(n).

The Space Complexity of ScheduleMaintainer is O(n).

ScheduleMaintainer is initialized by constructing an indexed (by External Time) multi-linked list data structure (EventList) that consists of records corresponding to every Event in the Universe of Events. Each of the n records (REj) include:

1. Name of the Event
2. Marker indicating whether the Event has occurred
3. Time stamp
4. Marker indicating whether the Time Stamp is observed, assigned as a constraint, or assigned arbitrarily by the program
5. Pointer to Predecessor REi
6. Pointer to Successor REk
7. Variable number of nonnull Pointers to other REs with quantitative constraint relationships between REi and the other individual REs
8. Corresponding quantitative constraint for each Pointer
9. Marker indicating whether the Record is to be Moved

The algorithm receives as input the name of an Event and an external time Stamp. The time stamp may be when the Event actually occurred or assigned by the user (to support interactive incremental rescheduling i.e. what-ifing).

The algorithm then examines the corresponding REi to determine if the REi should be moved in order to maintain a partially ordered (isomorphic) relationship between the discrete time of the time stamps of items on EventList and the natural numbers. This is done by comparing the new discrete time stamp with the time stamp of the successor RE.

If the new External time stamp violates the partial order condition, REi is marked to be moved and moved to a location that maintains the partial order condition.

In the new location, the successor to REi, REj is selected and a relation vector for the pair EiEj is generated. Depending on the time stamps of the two records, the vector is either = or >. If the time stamps are equal the vector is =. If the time stamps are ordered the vector is >.

The new relation Rij is then passed to ConstraintChecker.

FUTURE WORK

C++ is being used for the version currently under development. The new version of the prototype will provide an exploratory function which permits the user to query the system about the impact of changes to the preplanned countdown schedule. Both of the above developments are straightforward and will result in improved performance and increased functionality, respectively.

A more challenging task addresses the redundancy inherent in an array representation of the constraint set. We believe the bandwidth (e.g. Zabih, 1990). of the transitive closure of the countdown graph is quite small and substituting the transitive closure for the original graph, will permit us to profitably use an adjacency list (e.g. Mehlhorn, 1984) rather than an array representation of the constraint set. We currently believe we can maintain inferential soundness and completeness with such an approach. The issue seems to be, what impact this might have on the scope of the models specifiable with such a system. If we are able to use this approach, a substantial reduction in the time complexity of ConstraintChecker is possible.

REFERENCES

(Allen 83) James F. Allen, Maintaining Knowledge About Temporal Intervals, Communications of the ACM 26(11), 832-843, 1983

(Allen & Hayes 85) James F. Allen, Patrick J. Hayes, A Common-Sense Theory of Time, Proc. 9th IJCAI, Los Angeles (Cal.), 528-531, 1985

(Davis 87) Ernest Davis, Constraint Propagation with Interval Labels, Artificial Intelligence 32, 281-331,1987

(Mehlhorn 84) Kurt Mehlhorn, Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness, Springer-Verlag, 1984

(Schmiedel 90) Albrecht Schmiedel, A Temporal Terminological Logic, Proc 8th AAAI '90, Boston (Mass.), 1990

(Zabih 90) Ramin Zabih , Some Applications of Graph Bandwidth to Constraint Satisfaction Problems, Proc 8th AAAI '90, Boston (Mass.), 1990

(Vilain & Kautz 86) M. Vilain, H. Kautz, Constraint Propagation algorithms for Temporal Reasoning, Proc 4th AAAI '86, Philadelphia (Pa.), 1986