2014

# Real-time task attributes and temporal constraints

Amir Ashamalla
*University of Wollongong*, ana462@uowmail.edu.au

Ghassan Beydoun
*University of Wollongong*, beydoun@uow.edu.au

N Paramesh
*University of New South Wales*

## Publication Details

Ashamalla, A., Beydoun, G. & Paramesh, N. (2014). Real-time task attributes and temporal constraints. Americas Conference on Information Systems (AMCIS) 2014 Proceedings (pp. 1-11). United States: AIS Electronic Library.

# Real-time task attributes and temporal constraints

**Abstract**
Real-time tasks need attributes for monitoring their execution and performing recovery actions in case of failures. Temporal constraints are a class of real-time task attributes where the constraints relate the status of the task to temporal entities. Violating temporal constraints can produce consequences of unknown severity. This paper is part of our on-going research on real-time multi agent systems constraints. We discuss the importance of temporal constraints and present a task model that explicitly represents temporal constraints. We also present our preliminary results from our initial implementation in the domain of Meeting Schedules Management involving multiple users assisted by agents.

**Keywords**
task, attributes, temporal, time, constraints, real



**Publication Details**
Ashamalla, A., Beydoun, G. & Paramesh, N. (2014). Real-time task attributes and temporal constraints. Americas Conference on Information Systems (AMCIS) 2014 Proceedings (pp. 1-11). United States: AIS Electronic Library.

# RT Task Attributes and Temporal Constraints

*Submission Type: Research-in-Progress*

**A. Ashamalla, G. Beydoun, N. Paramesh**

*University of Wollongong*
*{ana462, beydoun} @uow.edu.au*
*University of New South Wales*
*paramesh@cse.unsw.edu.au*

## Abstract

Real-time tasks need attributes for monitoring their execution and performing recovery actions in case of failures. Temporal constraints are a class of real-time task attributes where the constraints relate the status of the task to temporal entities. Violating temporal constraints can produce consequences of unknown severity. This paper is part of our on-going research on real-time multi agent systems constraints. We discuss the importance of temporal constraints and present a task model that explicitly represents temporal constraints. We also present our preliminary results from our initial implementation in the domain of Meeting Schedules Management involving multiple users assisted by agents.

**Keywords**

Real time constraints, real time, multi agent systems, scheduling.

## Introduction

While we assume that Multi-Agent System (MAS) is a decentralized system, MAS reported in the earlier works are not necessarily decentralized. Existing multi-agent systems for reasoning with real-time constraints often use a central monitoring agent (master agent) (Neto et al 2009; Beydoun et al 2006) to achieve an overall synchronization. For example, the monitoring agent may initiate another task if an agent was not able to meet the real-time constraints of a task allocated to it in the previous problem solving cycle (Neto et al 2009). There are different kinds of constraints including temporal constraints, quantitative and qualitative constraints (Meiri 1996). A temporal constraint is "where [attribute] variables represent time and constraints represent sets of allowed temporal relations between them" (Schwalb and Vila 1998). In other words, temporal constraints can be viewed as constraints on the relative positions of tasks along the time line (Meiri 1996).

Constraints on real-time attributes of plans, actions, events and messages play a significant role in achieving the overall synchronization across the agents in MAS. For example, the London Underground project (Basra, Lu and Skobelev 2007) used temporal attributes of messages and actions taken by other trains to avoid collision. Other applications include cases such as search and rescue tasks (Micacchi and Cohen 2008) where temporal aspects of actions were used for avoiding obstacles in rescuing victims in real-time target tracking (Sabou, Faheem and Khalifa 2008), constructions (Zhang, Hammad and Bahnassi 2009), and automated car driving (Konrad 2006).

A *real-time agent* is an agent with temporal restrictions in its allocated responsibilities or tasks (Attoui 2000, Botti and Julian 2004). A real world task description is required to specify several temporal aspects of the activities that need to be taken into account while performing the task. For example, a plan may include temporal constraints on the sequence of actions, duration, deadlines and resource states. A temporal constraint on a set of entities is defined by a condition that must be satisfied by the entities over time. Temporal constraints are fundamental to the descriptions of real-time activities such as dialling a number to talk to a person, waiting for the call to be answered by a person, talking to the person and scheduling a meeting, and placing down the receiver. Activities both individually and collectively in this

example must satisfy the implied temporal constraints. For example, dialling a number must finish within a few seconds, and within a reasonable period of time the ring tone must be heard, etc., and the whole activity must finish within a pre calculated time. Violation of these constraints is often not accepted and it is necessary to specify them explicitly. In this paper, we discuss several types of temporal constraints incorporating them in an event based model of a task.

The rest of the paper is organised as follows. We first propose a set of real-time task attributes and discuss temporal constraints on tasks in the event plans. We then discuss application of temporal constraints in task execution management. We present some results from our preliminary implementation in the Implementation section and finally conclude the paper in the last section.

## Real-time Task Attributes and Temporal Constraints

Real-time constraints identify if/when a task is said to have failed to complete within an expected time frame. Figure 1 is a partial ontology of some of the core concepts from real-time task attributes, temporal constraints and the relationships between them. We believe identifying these attributes and constraints in the analysis phase in software development enables software engineers to better analyse and design an agent system.

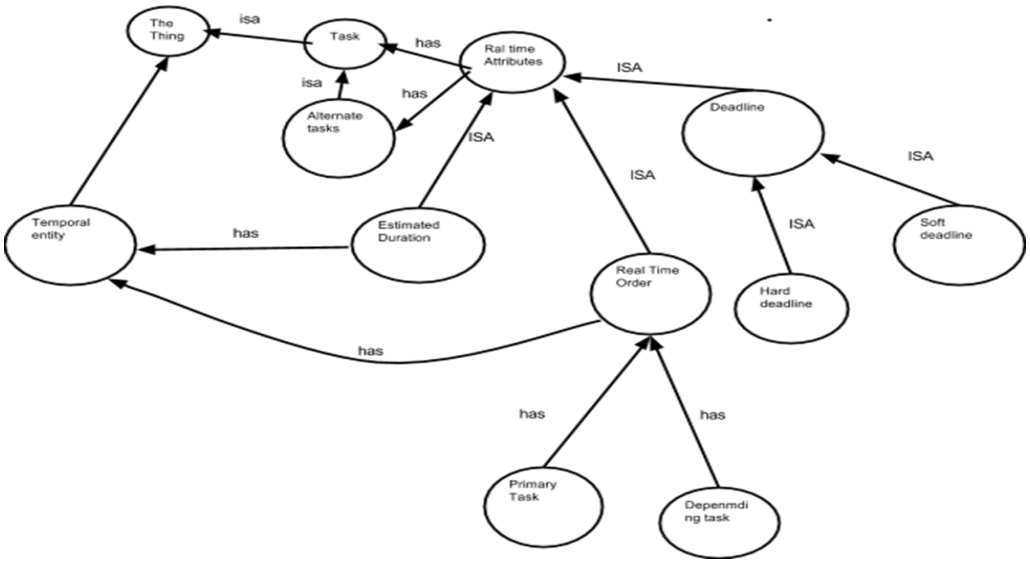


**Figure 1. A partial ontology of real-time task attributes**

## Task Model

We define temporal constraints on tasks by modelling tasks using events. An event $e_i$ is said to have occurred in W when the state $s_j$ of the world changes to $s_k$ and we denote such an event as $e_i$: $s_j \rightarrow s_k$. Figure 2 shows an event diagram in which several events have been drawn over a time line. In this paper, we distinguish three types of events: (a) an action event that is caused by the execution of an action $a_i$ by the agent in the world; (b) a non-action event $\lambda_i$ that occurs due to internal or external causes; and (c) a null event $\varepsilon$ that occurs when the agent chooses to execute a null action (also denoted by $\varepsilon$) whenever it does not want to execute any non-null action in the world. (Sometimes we use the notation e to denote any one of these events.) The execution of an action $a_i$ may also have a delayed consequence in the future in addition to an immediate consequence. In such situations, we model the delayed consequences using the $\lambda$ events. Events happen over time, and at some point in time the agent may see more than one option to choose from. Events and states may be related by relations such as causality and other domain dependent relations. Thus, in Figure 2, the execution of the action $a_0$ changes the state $s_0$ to $s_1$ at time $t=t_0$. At $t=t_1$, there is an option involving an action $a_1$, an event $\lambda_0$ and a null action $\varepsilon$. At $t=t_2$, there exists two options for the agent to choose from: $a_2$ and $a_3$ resulting in states $s_4$ and $s_9$, respectively. When the execution of $a_2$ fails (denoted as $\acute{a}_2$ at $t_2$) the world goes to the state $s_5$. At $t=t_3$ the execution of $a_5$ results in the state $s_7$ which also results when the execution of the action $a_4$ occurs at $t=t_4$. The states $s_6$

and $s_8$ are also labelled with the goals $G_0$ and $G_1$ signifying the fact that these goals are satisfied in these states. The event at $t_4$, $\lambda_2$ is the delayed consequence due to the execution of the action $a_2$ at $t_2$. Similarly, the state $s_3$ in the interval $<t_1, t_2>$ is a precondition for the action $a_6$ at $t_3$.

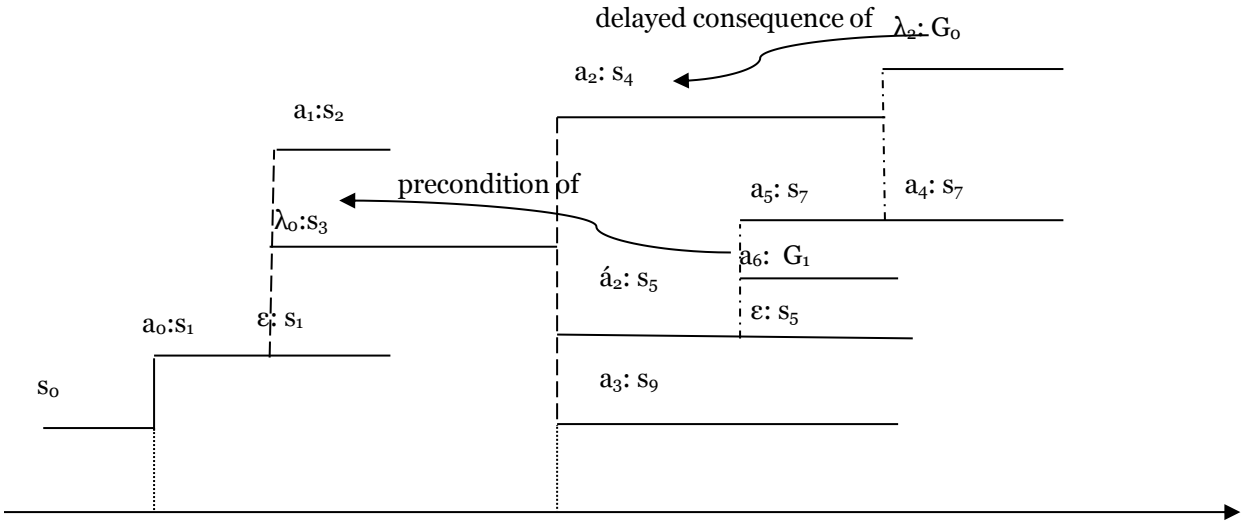


**Figure 2 Event Diagram**

## Task Attributes and Temporal Constraints

We have identified twenty three task attributes that are useful in monitoring the execution of real-time tasks.

1. **Alternate Tasks** (AT) A fundamental attribute that a real-time task has to have is Alternate Tasks. AT refers to the alternate options that an agent can consider at any point in time in case the current action fails. We define the degree of AT at a given point in time t for a task T, DAT(T,t), as the number of alternate task options the agent has at t. In Figure 2, when for example, the execution of $a_2$ at time $t_2$ fails, the world goes to state $s_5$ from where the agent can execute $a_6$ to complete an alternate (sub) task that has $G_1$ as its associated goal. AT is not a temporal constraint, but AT specifies alternate ways of achieving a task when constraints fail. Since real-time constraints are expected to be violated often in a dynamic world, a robust real-time task must preferably have as many alternate task options at each execution step. Absence of AT at any point in time, that is DAT(T,t)=0, signifies the fact that the task can fail at the point t irrecoverably.
2. **Deadline** Deadline refers to a point t on the timeline by which time the given task T execution must finish successfully. Deadline is often used to identify when a task has failed. Once the task T fails to meet its deadline, the agent will need to take an action (considering alternate tasks) depending on how critical the task is, for example, by notifying the affected agents (see Tier Number below). If the failing task is a core task (highly critical) of the system, the system may fail to run successfully or may crash. For some non-critical tasks, there may exist alternate tasks to run. For other non-critical tasks where no alternate tasks exist, the agent may have to restart/rerun ( see Retry attempts below) the same task once again or simply notify other dependent agents (Tier Number) that this task has failed. This is valid when the system can work without the failed task results/outcome. This is usually the case for non-critical tasks. The deadline for task T is defined as that point in time t which the agent is committed to, and where an action e occurs so to cause a state $s_2$ which the agent believes is the final state in the task execution and e is the last action in the event plan of the task. Thus,

   □ $s_1$ □ $s_2$ □ e □ t [ (Occurs(e,t) & (e: $s_1 \rightarrow s_2$) & Believe(FinalState($s_2$, T)) & LastEvent(e,T)) & CommitToExecute(e,t) )→ Deadline(T,t)]

   CommitToExecute( ) denotes commitment of the agent to execute e at time t. We associate with a task

T the notion of utility of the task T achieved at or before time t , U(T,t) ϵ R where R is the set of real numbers, and it denotes how significant the completion of T at or before time t is to the overall health of the system.

a. **Hard Deadline** A hard real-time deadline enforces that a task T be completed within a specified time $t_o$ and in such a case $U(T, t_o) = 1$ ; otherwise the outcome of the task is unacceptable or is of no value, and $U(T, t_o) = 0$.
b. **Soft Deadline** Sometimes, a task T is allowed to be completed even after the deadline $t_o$ expires, but in such cases, the utility value of such a completion may be considered to be declining for all $t > t_o$ ; that is, a task completed after its deadline is considered to be of less utility value to the overall system than the task whose deadline is yet to expire. Critical tasks normally have hard constraints. However, not all hard real-time constraint tasks are considered critical. For example, in a PhD program, publishing a paper would have a hard constraint (paper submission date). However, paper publication is not a critical requirement for completing a PhD thesis, whereas the PhD end date, first year progress review, annual progress reports, etc. may all be considered as critical tasks with hard constraints. Thus,
□ $t_1$ [HardDeadline(T,$t_1$) & ( □e□ $s_1$ □$s_2$ □$t_2$ Occurs(e, $t_2$) & (e:$s_1 \rightarrow s_2$) & Believe(FinalState($s_2$) ) & $t_2 > t_1$ ) → □ $t_3$ ($t_3 > t_1$ → U(T, $t_3$) = 0 ] where T is a task.

Similarly, we can define a soft deadline of task T to be another time point $t_1 > t_o$ where $DAT(T,t_o) > 0$, $t_1 - t_o \leq k_{max}$ and $U(T,t) <= U(T,t_o)$ for all $t > t_o$ and for some constant $k_{max}$; that is, the task T has the option to finish by $t_1$ but possibly with a lower utility value. Thus,

□ $t_1$ [Deadline(T,$t_1$) & □e□ $s_1$ □$s_2$ □$t_2$ ((Occurs(e, $t_2$) & (e:$s_1 \rightarrow s_2$) & Believe(FinalState($s_2$) ) & $t_2 - t_1 \leq k_{max}$ ) & □ $t_3$ ( $t_1 < t_3 < t_2$ → U(T, $t_3$) > 0 ) → SoftDeadline(T,$t_1$) ] where $k_{max}$ is a positive constant.

3. **Estimated Duration** (ED) Estimated Duration of a task T, ED(T), of an agent denotes the time that the agent estimates to complete the task, and it is an important temporal attribute of a task. While defining ED, we assume that there exists a sequence of options of events starting from the current state $s_i$ at time $t_i$ to a final state $s_f$ at time $t_f$. Then, $ED = t_i - t_f$. The occurrence of external events λ affects the value of ED. In the absence of λ events, it is possible to get a reasonable estimate of the ED for a given task. However, in the presence of λ events, ED is estimated from the historical runs of that task. Since the first run or instance of the task will not have any history, the initial estimate of ED(T) is often based on the software engineer's input and/or lines of code. A temporal constraint placed upon a task using ED will typically require that ED of a task lie below a specified limit. For example, $ED(T) \leq n_{max}$ is represented as an assertion below :
□ $e_1$ □ $e_2$ □ $s_1$ □ $s_2$ □ $t_1$ □ $t_2$ [Occurs($e_1$,$t_1$) & Occurs($e_2$,$t_2$) & [$e_2$:$s_1 \rightarrow s_2$ & Believe(FinalState($s_2$))]→ $t_2 - t_1 \leq n_{max}$ ] /* Agent believes $s_2$ is final state in the task execution*/

4. **Real-time Order** (RTO) Real-time Order of a task $T_1$ is a temporal attribute of the task which is defined as the time between the finishing of $T_1$ and a dependent task $T_2$ starting. If $T_1$ finishes at time $t_1$ and $T_2$ starts at $t_2$ then $RTO = t_2 - t_1$. When RTO is positive it is called slack time and when negative it is called leap time. The more slack time a task has the less priority it would have, as it will have more time to be delayed without affecting or missing its deadline. Constraints are placed on the values of RTO. For example, a typical constraint of this type will be of the form: $RTO \leq n_{max}$. Thus,
□ $e_1$ □ $e_2$ □ $t_1$ □ $t_2$ [Occurs($e_1$,$t_1$) & Occurs($e_2$,$t_2$) & $t_2 - t_1 \leq n_{max}$ & Believe(LastEvent($T_1$,$e_1$)) & Believe(FirstEvent($T_2$,$e_2$)) → Starts($e_1$,$e_2$,$t_2$) ] /* $e_1$ starts $e_2$ at $t_2$ where $e_1$ is the first event of $T_1$ and $e_2$ is the last event of $T_2$ */
Deadline is a function of RTO. If RTO > 0, then the task duration can be extended by that value before the dependent task fails and vice versa.

5. **Periodic Occurrence** (PO) Periodic Occurrence of a task T constraints the task to occur periodically. (See Figure 3.) The implication of periodicity is that the task T will keep recurring with a periodicity P over the specified interval of time L.

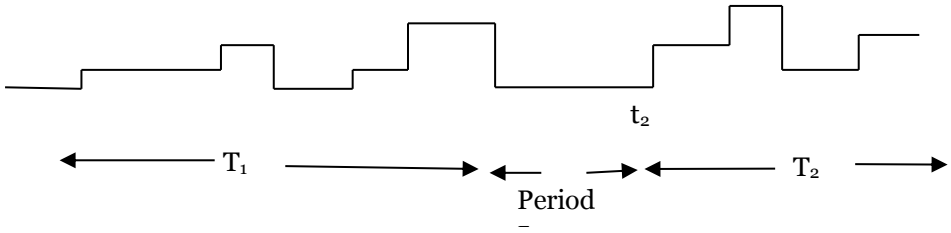


**Figure 3 Periodic Task T where T1 is its first occurrence and T2 its second occurrence.**

If $T_1$ and $T_2$ are two adjacent task instances of a task T then: $\forall e_1 \forall e_2 \forall t_1 \forall t_2$ [Occurs($e_1$,$t_1$) & LastEvent($e_1$,$T_1$) & Occurs($e_2$,$t_2$) & FirstEvent($e_2$,$T_2$) → ($t_2 = t_1+\tau$)] for some constant τ called the period. In a real world scenario, a periodic task signifies the fact that it occurs more than once, and in certain situations if the agent fails to achieve it in a given occurrence, the agent may look forward to retrying it at a later occurrence. (The Retry attempt is a function of PO as stated below.)

6. **Sampling Time** (ST) Sampling Time is a temporal attribute of a real-time task that identifies the status of a task at chosen time instants. In Figure 4, we have shown two rates of sampling Sampling1 and Sampling2 where sampling has been modeled as events of sampling of the state of task execution.

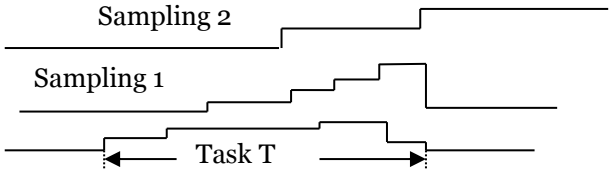


**Figure 4 Sampling time**

The rate of sampling is a function of how critical the task is. A critical task should be sampled more often than a non-critical task to identify early potential delays and the delays must be minimized by, for example, choosing appropriate alternate tasks and assigning more resources. There is an upper limit of sampling times as sampling by itself consumes resources and could potentially delay tasks rather than help resolve task conflicts and/or delays. Thus, $\forall n\,(T,n) \leq n_{max}$ , where n is an integer, for some maximum value of sampling of a task T. Sampling can identify the task status as one of the three: on track, delayed, and early. Agents will take actions based on the task status, dependent tasks status and the system. If the task is on track, then no action is required. If the task is early the agent might need to delay it, for example, by providing other delayed task more resources to catch up and become on track. In another scenario when an early task has a dependency that will not complete early, we delay this task for its results to be usable on time by the dependent task.

7. **Priority** (P) Priority on a set of tasks is a temporal attribute that specifies the temporal order on the tasks that have to be executed. It is a function of the status of the tasks and how critical each one is. A critical task running late should have a higher priority than a non-critical task running ahead of schedule. All critical tasks should have a higher priority than non-critical tasks. Priority is also a function of the periodic occurrence of a task where if a task occurs in very low intervals then its priority decreases and if it fails then the next instance of that task will run relatively sooner without any need for the agent's interference, rescheduling or re-planning. The more slack time a task has the less priority it would have, as it will have more time to be delayed without affecting or missing its deadline.

8. **Retry** Retry n is a real-time non temporal task attribute that indicates the number of times n a task may be re-attempted whenever it fails. It is assumed that Retry options exist whenever the task fails and the agent wants to retry the task. Retry attempts is a function of AT and PO. If the PO is low (say, every one millisecond) then there is no need to retry as the task will automatically run. However, if it keeps failing n times on a schedule or on a Retry attempt, then this task is considered broken and the agent would consider alternate tasks.

9. **Criticality** (C) Criticality is a non temporal task attribute that indicates how critical a task is and it signifies the effect of failure of the task would have on the whole system. The degree of criticality is a function of dependent and alternate tasks. A task may become critical as the number of its dependent tasks increases and the number of its alternate tasks decreases. For example, a task $T_1$ with no alternatives and several other tasks depending on $T_1$ would be considered highly critical, while a task $T_2$ that has several alternate tasks and no other tasks relying on its outcome would be considered less critical.

10. **Other Task Attributes** We also have identified several other task attributes which we mention below briefly due to lack of space.
    a. **Tier Number** (TN) Tier number of a task T denotes the total number of tasks depending on T. The criticality degree we presented above is a function of the Tier number of a task.

    b. **Max Output Jitter** (MOJ) Max Output Jitter is the difference between the best execution time and the worst execution time.

    c. **Task Status** (TS) TS represents the current state of the task which is one of: started, working, super, and final.

    d. **Check Points** (CP) CP represents a point where task results can be saved. The assumption is that software faults can be overcome by re-execution of the affected task from the most recent checkpoint.

    e. **Validity Duration** (VD) This refers to the maximum time the data can be held before expiring or being considered invalid.

    f. **Slack Time** (ST) ST refers to the time within which the execution time can be increased without failing the deadline.

    g. **Minimum Time** (MT) Minimum time is the minimum time required for a task to complete.

    h. **Instant Value Function** (IVF) IVF refers to the total accrued value of a job.

    i. **Execution Accrued Value** (EAV) EAV measures the amount of value gained by the system, in terms of time gained due to tasks completing below their deadlines and/or estimated duration.

    j. **Real or Not** (R/N) R/N identifies if the task is a real time component or not.

    k. **Remaining Time** (REM) REM identifies the remaining time till the deadline is reached.

    l. **Maximum-Miss-Ratio** (MMR) MMR is a soft deadline that cannot be missed more often than a specified number of times or ratio.

m. **Composite** (COMP) COMP refers to a list of simple timing requirements that are imposed at the same time; hence the constraint is the composite time requirement of the individual requirements.

## Event based Modeling of Rules

Rules are useful in specifying how task execution can be monitored and task repairs can be performed. A rule has a condition that should be satisfied to perform an action. Temporal constraints can occur both in the conditions and in the actions. Rules can be modeled using the event notations we described above. In Figure 5, we have considered a rule r where

*r:* $C_1$ $(s_1)$ & $C_2$ $(s_3)$ & $(t_2\text{-}t_1 \leq t_3\text{-}t_2) \rightarrow$ *select_execute* $(a_0, t_4)$.

The rule above uses events semantics (Yao-Hua and Thoen 2002) and reads as: if states $s_1$ and $s_3$ satisfy the conditions $C_1$ and $C_2$ respectively and the temporal constraint $(t_2\text{-}t_1 \leq t_3\text{-}t_2)$ is also satisfied, then select the action option $a_0$ and execute it at time $t_4$. In Figure 5, the events $e_1$, $e_2$, and $e_3$ cause the states $s_1$, $s_2$ and $s_3$. We graphically show that the state $s_4$ , which was caused by the execution of the action $a_0$, is supported by the states that occurred in the past.

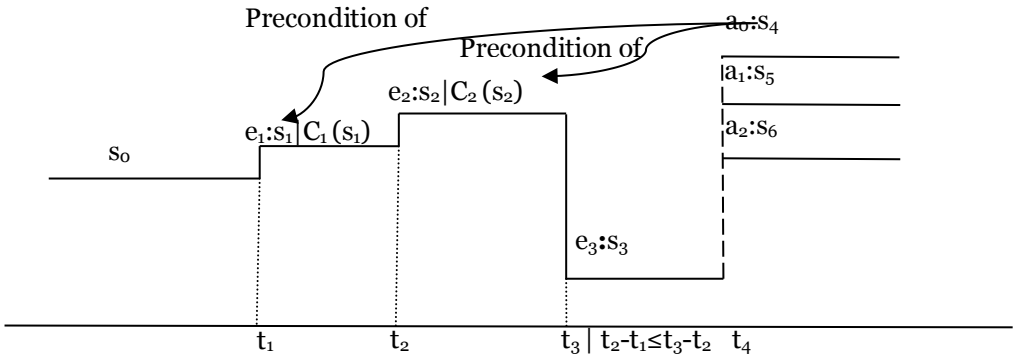


**Figure 5 Modeling rules**

## APPLICATION TO TASK MANAGEMENT

In real world situations, task management involves executing actions and, when actions fail, attempting recovery actions. Event plans need to have more execution recovery strategies built into them than the traditional action plans. Figure 6 shows a simple strategy for event plan execution.

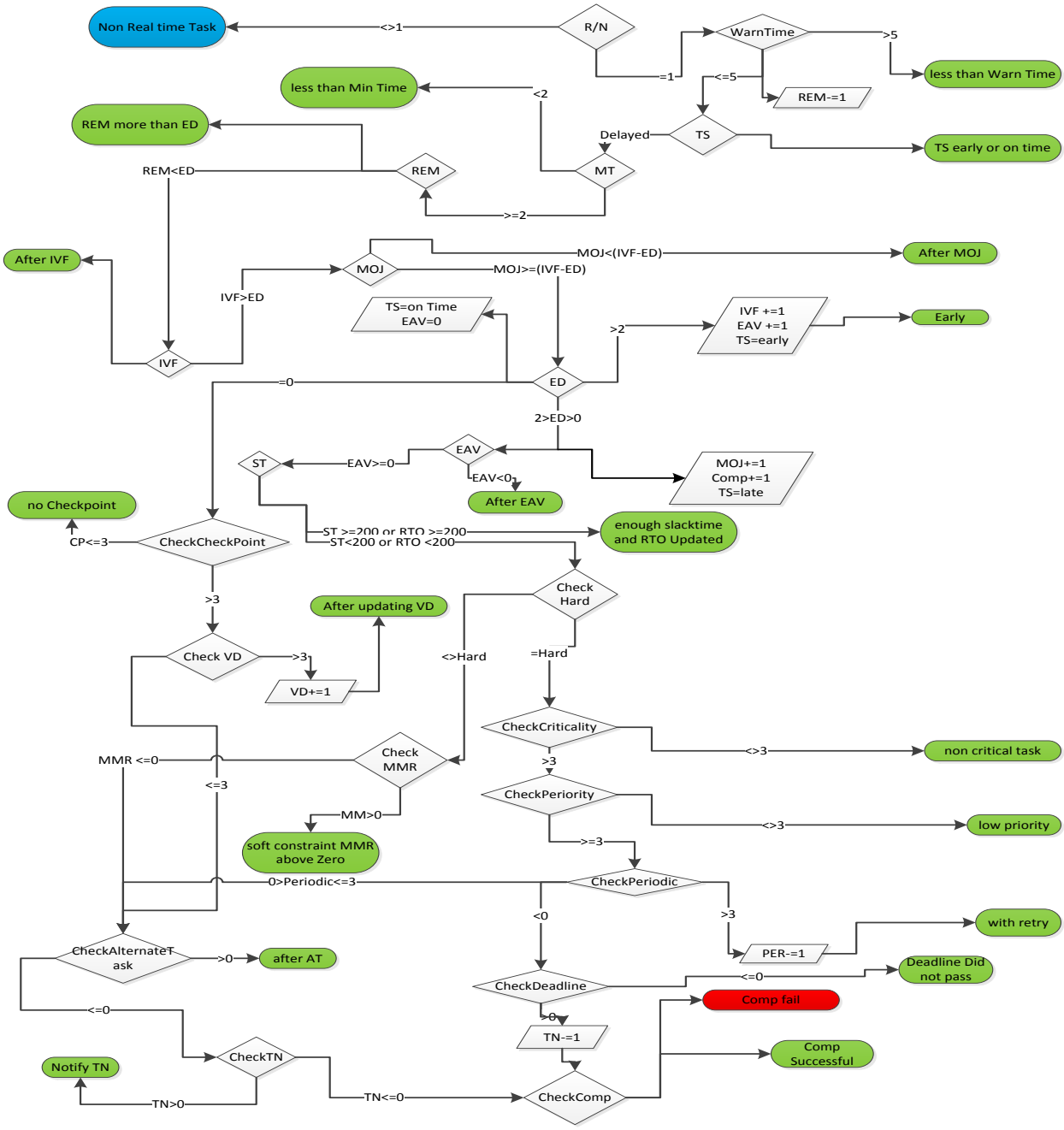


**Figure 6. Event plan execution and constraint violation recovery.**

# IMPLEMENTATION A TEMPORAL CONSTRAINT RESOLVER

Temporal Constraint Resolver (TCR) is a system that we implemented for Meeting Schedule Management. This is a multiagent system (running on Android phones) which organizes several meetings as an event plan for its users. Typical goals include meeting at a chosen point in time by several agents where each agent has multiple options to travel, namely, by walk, bus or train. A meeting is said to have

failed if at least one of the agents that was committed to the meeting is not able to attend the meeting. This can occur if an agent fails to take the trip by the planned means. The agent computes estimated duration, task state by choosing a sampling rate, etc. The agent then attempts other options which results in the deviation.

We have developed an application on the Windows platform that uses a calendar on a mobile device using code from (Dellai 2013) currently incorporating: (a) twelve real-time constraints; and (b) eighteen real-time constraints. The experiments demonstrated that adding real-time constraints has actually improved robustness of the application and the scheduling. The user was notified when he was running late for appointments, giving him enough time to reschedule his meeting or choose other alternate faster traveling methods to help him arrive on time. The twelve constraints were not considered a sufficient set (Ashamalla, Beydoun and Low 2009); however using the eighteen constraints were sufficient and was preferred than using the twenty three constraints identified in our earlier work (Ashamalla, Beydoun and Low 2012). The eighteen constraints accurately identify delayed tasks. For example, when using the twelve constraints, the slack time was compared to the delay, which in certain cases did not give accurate results as compared to the EAV (Execution Accrued Value - the time gained due to tasks completing before their deadlines and/or estimated duration); that is, gained time enables delays in a task not affecting dependant tasks as long as the delay is less than the gained time. The implemented Multi agent system was based on multicasting (Microsoft team 2013). We measured the utilization of cpu, disk, and network in our implementation for the cases: (a) no constraints; (b) 18 constraints; and (c) 12 constraints.

1. CPU - Processor Time: This counter provides a measure of how much time the processor actually spends working on productive threads and how often it was busy servicing requests.
2. Disk - Current Disk Queue Length: This counter provides a primary measure of disk congestion. The disk queue is an indication of the number of transactions that are waiting to be processed.
3. Memory - Committed Bytes in Use: This counter indicates the total amount of memory that has been committed for the exclusive use of any of the services or processes on Windows
4. Network - Bytes Total/Sec utilization: This indicates how much information is going in and out of the interface.

Due to shortage of space, we only show partial network utilization results in Figure 7.

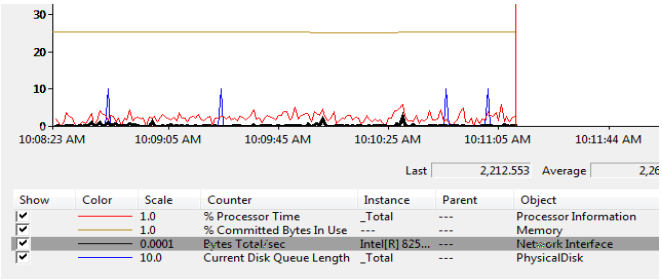


Figure 7 (a)

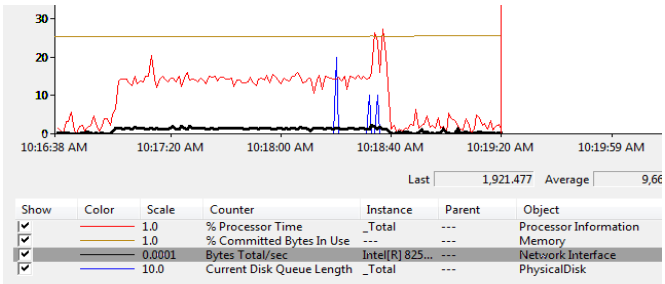


Figure 7 (b)

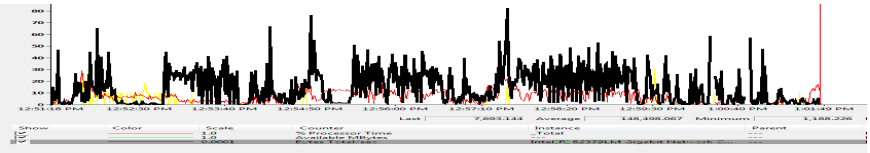

Figure 7 (c)

**Figure 7 Network utilization for (a) no constraints, (b) 18 constraints, and (c) 12 constraints.**

| | 0 | | 12 | | 18 | |
|---|---|---|---|---|---|---|
| | **Max** | **Average** | **Max** | **Average** | **Max** | **Average** |
| **CPU** | 5.81 | 2.25 | 15.95 | 9.87 | 27.46 | 11.18 |
| **Disk** | 1 | 0.022 | 0 | 0 | 2 | 0.029 |
| **Memory** | 25.52 | 25.39 | 25.56 | 25.37 | 26.65 | 25.49 |
| **Network** | 25,516 | 2,261 | 19,331 | 9,071 | 21,593 | 9,668 |

**Table 1 Load test results for the cases: (a) no constraints, (b) 18 constraints, and (c) 12 constraints**

Table 1 illustrates the difference between processing 18 constraints and processing only 12 constraints. It is seen that the difference is very small as the average CPU went from 9.87% to 11.18%, while adding only 6 constraints. But adding the initial 12 constraints increased the CPU utilization from 2.25% to 9.87%.

## Conclusion

Assistance offered by our system when the users ran into delay has proved to be very effective, since delays were communicated to other meeting members and if the person was running too late, the meeting then was rescheduled to another time. In case an alternate meeting time was available, that was the one directly rescheduled to. If no alternate meeting time was available a bidding processes was started (to identify a new goal $G_i$) and the highest meeting bid was chosen. The bidding processes were a simple bidding where each user selected one time slot and the time slot with the highest bid was chosen. This research did not focus on bidding algorithm or methods as it was out of our research scope. Agents were aware of each other and managed to identify when other agents went offline. However with physical mobiles when the agent went offline, it did not mean the person will not attend the meeting as mobile batteries could have run out or the mobile would be out of coverage, etc. In other domains this would have to be considered since if it were tasks running on other computers and if the computer went down then all its tasks would fail, unless it has a self-healing or redundancy mechanism. This should be considered and identified when designing the application.

The scheduling problem was chosen as it can be mapped to other problems without loss of generality of the results obtained. Furthermore, nearly ever reader of this research would have experienced being late for a meeting or deadline so knowledge of this problem is quite common. The mapping of results to other domains can follow ontology mappings as outlined in (Tran, Beydoun and Low 2007; Drake and Beydoun 2000). Towards this, a call center case study which illustrates this mapping is being developed; however due to size restrictions it was not presented in this paper.

This paper is part of our on-going research aimed at identifying and modelling real-time constraints in the early analysis stage of the development life cycle. The model would help developers, analysts and

researchers identify and avoid future bottlenecks where agents are being overloaded with real-time constraints. The model also helps illustrate, model and understand system real-time constraints.

## REFERENCES


Bernat, G., A. Burns, et al. (2001). Weakly hard real-time systems. Computers, IEEE Transactions on 50(4): 308-321.

Botti, V. and Julian, V. (2004). Developing real-time multi-agent systems. Integrated Computer-Aided Engineering 11(2): 135-149.

Ashamalla, A., Beydoun, G. and Low, G. (2009). Agent Oriented Approach to a Call Management System, 18th International Conference on Information Systems Development (ISD 2009), Nanchang, China, September 16-19

Ashamalla, A., Beydoun, G. and Low, G. (2012). Towards Modelling Real-time Constraints , 7th International Conference on Software Paradigm Trends (ICSoft 2012), Rome, Italy, July 24-27

Neto, A., Sartori, F., Piccolo, F., Vitelli, R., De Tommasi, G., Zabeo, L., Barbalace, A., Fernandes, H., Valcárcel, F. and Batista, N. (2009). MARTe: a Multi-Platform Real-Time Framework. Proc. of the 16th IEEE NPSS Real-Time Conference, Beijing, China.

Beydoun, G., Gonzalez-Perez, C., et al. (2006). "Developing and Evaluating a Generic Metamodel for MAS Work Products". Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications. A. Garcia, R. Choren, C. Lucenaet al. Berlin, Springer-Verlag. LNCS 3914: 126-142.

Beydoun, G., Tran, N., Low, G. and Henderson-Sellers, B. (2006) Foundations of Ontology-Based Methodologies for Multi-agent Systems. *Proceedings of AOIS2005* (eds. M. Kolp, P. Bresciani, B. Henderson-Sellers and M. Winikoff), LNAI 3529, Springer-Verlag, Berlin, 111-123

Beydoun G and Hoffmann A (1998). Simultaneous Modelling and Knowledge Acquisition using NRDR. 5th Pacific Rim Conference on Artificial Intelligence (PRICAI98), Singapore, Springer-Verlag.

Basra, R., Lu, K. and Skobelev, P. (2007). Resolving scheduling issues of the London Underground using a multi-agent system. International Journal of Intelligent Systems Technologies and Applications 2(1): 3-19.

Drake, B. and Beydoun, G. (2000). Predicate logic-based incremental knowledge acquisition. In P. Compton, A. Hoffmann, H. Motoda and T. Yamaguchi (Eds.) *Proceedings of the sixth Pacific Knowledge Acquisition Workshop (PKAW 2000)*, Sydney, Australia, 71-88.

Micacchi, C. and Cohen, R. (2008). A framework for simulating real-time multi-agent systems. Knowledge and Information Systems 17(2): 135-166.

Sabour A., Faheem M. and Khalifa E. (2008). Multi-Agent Based Framework for Target Tracking Using a Real-Time Vision System. International Conference on Computer Engineering and Systems, ICCES 2008 355-363

Zhang, C., Hammad, A. and Bahnassi, H. (2009). Collaborative Multi-Agent Systems for Construction Equipment Based on Real-Time Field Data Capturing. Electronic Journal of Information Technology in Construction 14: 204-22.

Konrad, J. (2006). Model-driven development and analysis of high assurance systems. Department of Computer Science. Michigan, Michigan State University. DOCTOR OF PHILOSOPHY: 443.

Attoui (2000), A. Real-Time and Multi-Agent Systems, 1st edition, Springer- ISBN: 1-85233-252-2.

Yao-Hua, T. and W. Thoen (2002). Using event semantics for modeling contracts. System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on System Sciences.

Kakkad, J. and N. Parameswaran (2012). Efficiency Considerations in Policy Based Management in Resource Constrained Devices. Advances in Grid and Pervasive Computing. R. Li, J. Cao and J. Bourgeois, Springer Berlin Heidelberg. 7296: 210-220.

Meiri, I. (1996). "Combining qualitative and quantitative constraints in temporal reasoning." Artificial Intelligence 87(1–2): 343-385.

Schwalb, E. and L. Vila (1998). "Temporal Constraints: A Survey." Constraints 3(2-3): 129-149.

Tran, Q.N.N., Beydoun, G. and, Low, G. (2007). "Design of a peer-to-peer information sharing MAS using MOBMAS (ontology-centric agent oriented methodology". In Advances in Information Systems Development, Springer pp. 63-76.