

Co-ordinating Heterogeneous Interactions in Systems Composed of Active Human and Agent Societies

Konstantinos Prouskas and Jeremy Pitt

Intelligent and Interactive Systems Group
Department of Electrical and Electronic Engineering
Imperial College of Science, Technology and Medicine
Exhibition Road, London SW7 2BT, U.K.
+44 (0)20 7594 6318
`{k.prouskas,j.pitt}@ic.ac.uk`

Abstract This paper describes the specification and implementation of the middle layer in a new three-layer time-aware agent architecture. This architecture is designed for applications and environments where societies of humans and agents play equally active roles, but interact and operate in completely different time frames. The middle layer, called the Time-Aware Layer, uses services of the underlying real-time layer to co-ordinate the heterogeneous interactions present in composite human-agent systems. Interactions are unified by abstracting away from their temporal representation, temporal scale and class of parties they involve (be they humans or agents). To achieve this, this paper firstly introduces Availability Functions as the primary mechanism of reasoning about temporal constraints placed on interactions. It subsequently describes their stylised analytic representation and develops a Selective Sampling Algorithm which allows searching through them in bounded time. The resultant implementation allows more effective engineering of the topmost application layer firstly by providing an abstract, unified view of interactions and secondly by predicting and guaranteeing their initiation and completion times.

1 Introduction

Many agent systems operating in the real world affect or depend on humans for their correct and effective operation. In such systems, agents have temporally constrained goals and need to interact with other *agents* as well as *humans* in complex patterns in order to achieve them. Their temporal behaviour is determined by the set of temporal constraints placed equally on the computations and the interactions within the system.

In this paper we describe the specification and implementation of the middle layer in a new three-layer *time-aware* agent architecture. Time-aware agents are defined as agents capable of dealing with the hard, fast temporal dimension of agent-to-agent interactions while equally handling the much softer and slower

interactions with humans. This architecture is designed for applications and environments where societies of humans and agents play equally active roles, but interact and operate in completely different time frames.

We focus on the architecture’s Time-Aware Layer, which uses services of the underlying real-time layer to co-ordinate the heterogeneous interactions present in composite human-agent systems. To facilitate this co-ordination, interactions are unified by abstracting away from their temporal representation, temporal scale and class of parties they involve (be they humans or agents).

To achieve this, we firstly introduce Availability Functions as the primary mechanism of reasoning about temporal constraints placed on interactions. We subsequently describe their stylised analytic representation within the layer and develop a Selective Sampling Algorithm which allows searching through them in bounded time.

Section 2 provides background information and describes our motivation for this work. Section 3 presents an overview of the overall time-aware architecture and Section 5 focuses on its Time-Aware Layer in which interaction co-ordination takes place. Section 6 performs a brief evaluation before conclusions are presented in Section 7.

2 Background & Motivation

2.1 Composite Human-Agent Systems

Practical use of agent systems has seen their application in areas where there is extensive interaction with humans at the most basic level and societal or organisational structures more generally. Interactions between agents and humans play a key role in areas such as agent-enhanced workflow and e-commerce.

Such systems comprise two classes of entities: agents and humans, each organised in their respective society (Figure 1). Normally, the agent society is formed by a software multi-agent system, while the human one reflects the roles and relationships of the ‘real’ human societal structure.

Each society is not passive and does not function independently from the other, but forms an active and integral element of the environment. As an example of this, consider the following envisaged agent scenario [2]:

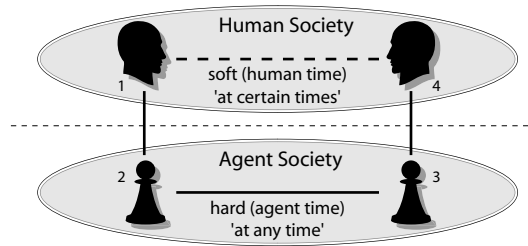


Figure 1. Composite systems consisting of agent and human societies.

The scenario takes place at a doctor's office in which a handheld agent is asked to produce a schedule for a physiotherapy treatment. The agent engages in conversation with the doctor's agent to retrieve the prescribed treatment and subsequently contacts a series of provider agents to determine the best one. After a few minutes the agent returns a proposed schedule of treatment to its human owner. Unfortunately, the schedule is unsatisfactory because it involves the owner driving across town during rush hour. Another agent is then given a stricter set of constraints and asked to come up with an alternative schedule. The second agent returns a better-suited schedule, albeit one that requires its human owner to cancel some of their less important appointments.

The system is initially unsuccessful in its attempt to provide a solution because it has not taken into account that the affected party is a human and therefore bound by a wholly different set of temporal constraints (in this case, the concept of rush hour) than a software agent.

Furthermore, there is unnecessary iteration because humans are never treated as active parts of the system. Instead, they are external, passive elements: the system is a black box in which humans feed some input and receive some output. This results in a trial-and-error approach: problems are not detected until the results are produced, at which point a human has to make modifications and retry.

To avoid these limitations it is therefore desirable to treat both humans and agents as active parts of these systems.

2.2 Co-ordinating Heterogeneous Interactions

The challenge of making agents and humans equally active participants lies in the fundamentally different temporal nature of interactions that it would encompass: agent-agent interactions (within the agent society), human-human interactions (within the human society), and also human-agent interactions in both directions (originating at either society).

From a temporal viewpoint, agent-agent interactions are well defined, fast and predictable, while human-agent interactions are very loosely defined and constrained and operate on an entirely different temporal scale. Human-human interactions are even more loosely defined as they introduce human uncertainty at both ends.

Agent-agent interactions normally assume availability of both parties (they can occur 'at any time') and response times in the order of seconds at most. Additionally, they are commonly iterative in nature and follow to a greater or lesser degree some protocol convention. Human-agent interactions are constrained either by physical location limitations, as when a human is 'out of reach' of an agent, or by higher-level concepts such as attention, interest, mood etc. (that is, they only occur 'at certain times'). Further factors that may affect human-agent interactions include day/night-time, time of day, working hours, bank holidays, illness, social commitments, biological reasons etc. Human-human interactions are a generalisation of human-agent interactions where both parties are human, and are the least predictable of the three.

This discrepancy stems partially from the difference in which agents and humans understand and deal with time. Agents, being principally software computing entities, have an implementation-level representation of time such as vector clocks commonly found in distributed systems, whereas humans are used to dealing with abstract relative concepts such as ‘today’ and ‘tomorrow’. Agents also have a much finer-grain representation of time like ‘seconds elapsed since January 1st, 1970’, whereas humans normally only need a much coarser granularity like ‘3 o’clock’ or ‘this afternoon’. Finally, agents have well-defined means of measuring temporal intervals (by measuring the difference of two UTC-synchronised clocks, for example), whereas humans rely on a variety of calendar systems and intervals of variable duration such as ‘evening’, ‘month’, ‘year’ and so on.

2.3 Related Work

There are several examples of *real-time agent architectures*, both ‘hard’ [6,3] and ‘soft’ [10]. Real-time agent systems are systems in which agents are temporally constrained in achieving their goals. They combine real-time (RT) and artificial intelligence (AI) characteristics by either embedding AI in RT, embedding RT in AI, or by adopting a co-operative RT-AI approach [7].

However, applying these architectures to this particular problem presents two key limitations. The first is that they mainly consider systems comprised of a single class of entities, namely software agents. Adding a human class, however, is problematic as humans ‘are not real-time’, that is, they cannot be considered to be the well-defined, predictable, principally computational entities their agent counterparts are.

The second limitation is that these architectures assume the internal deliberative process of an agent is exposed and focus on temporally constraining it in real time. In extending this to the type of composite human-agent systems we are considering, however, expressing temporal constraints on the deliberative process of a human actor is not appropriate. This is similar to deontic specifications in software engineering, where placing obligations on human actors is equally inappropriate.

On the other hand, work on *collaborative systems* between agents and humans either does not address temporal constraints [1] or, when it does [8], it is not from a real-time, predictability perspective. There are also examples of “dramatic failures” in such systems [4] because the temporal distinction between humans and agents has not been made explicit in the design.

There is therefore a need to identify the different classes of parties present; secondly to reason about how their interactions (rather than their deliberative processes) affect the system’s temporal behaviour; and finally be able to handle all interactions in a uniform manner, irrespective of the parties they involve.

3 A Time-Aware Agent Architecture

We have developed a real-time architecture, comprising *time-aware* agents, that better meets the requirements described above. Time-aware agents are defined

as agents capable of dealing with the hard, fast temporal dimension of agent-to-agent interactions while equally handling the much softer and slower interactions with humans. Time-aware agent systems can deal with an amalgam of hard, soft, human and non-real-time interactions, reason about the temporal constraints placed on the system by each type of interaction, make transformations between them and co-ordinate (schedule) activities seamlessly irrespective of their constituent constraints.

In this architecture, we consider systems comprised of a number of *parties*, agents or humans, logically or physically distributed over an interconnected network environment such as the Internet. Agents are lightweight computational entities that cannot share any data except by means of interaction through message passing. They are trustworthy and work co-operatively toward achieving the system's goals.

We primarily consider the co-ordination of agent-agent and human-agent interactions. Human-human interactions are considered only as far as they are agent-mediated, that is, emergent from a specific pattern of interaction (namely $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ in Figure 1).

We treat deliberation as internal to both agents and humans, treating both opaquely. We instead focus on providing the environmental and architectural mechanisms by which agents can be aware of the different types of interactions they are engaged in, as well as their temporal repercussions in achieving their goals.

We have realised this architecture using a prototype implementation in the Agent Process Interaction Language (April) [5], consisting of three layers: the *April Real-Time Run-Time* layer, the *Time-Aware Layer* (TAL) and the *Application Agent Layer* (AAL). The ART layer forms the underlying real-time agent platform. Similar to a real-time operating system, the ART provides temporal guarantees to the rest of the architecture. More information on the ART can be found in [9]. The AAL is the overlying application layer; in it, software agents exist and work toward their goals. The discussion in this paper focuses on the middle of the three layers: the Time-Aware Layer.

4 The Time-Aware Layer

The Time-Aware Layer (TAL) is the layer that provides a unified temporal view of the entire system. The TAL abstracts away from different time representations, disparate temporal scales and class of parties (human or agent) engaged in interactions. The main components of the TAL are (Figure 2):

Human/agent handles: Both humans and agents are uniquely identified by their *handle*. Handles are an abstraction mechanism by which parties in the system can be uniformly addressed.

Task representation: Tasks are hierarchically decomposed sequences of either *processing* or *interacting* (communicating).

Unified Time Representation: The Unified Time Representation (UTR) is an April macro layer responsible for transforming different human and agent

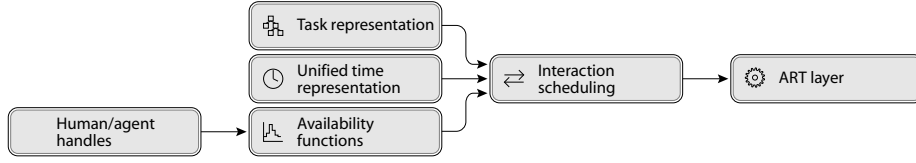


Figure 2. The Time-Aware Layer.

time representations, units, and calendars into a common representation (namely, Universal Coordinated Time). Human concepts like time zones, working hours, next working day, etc. are handled by the UTR.

Availability Functions: Availability Functions are associated with each party to express that party’s degree of presence within the system. They are fully described in Section 5.1.

Interaction Scheduling: Interaction scheduling involves co-ordinating and guaranteeing initiation and completion times of interaction sequences (generated from tasks). Interaction scheduling is detailed in Section 5.2. Schedules generated by this process are passed on to the underlying ART layer.

5 Realisation

Interaction scheduling is central to the TAL and Availability Functions play a key role in this process. In order to make interaction scheduling fast and predictable enough for real-time application, we constrain the Availability Function representation and discuss a Selective Sampling Algorithm.

5.1 Availability Functions

The TAL uses the concept of *availability* and *Availability Functions* (AFs) to reason about the temporal nature of interactions. The availability of a party i at time t , given by $V_i(t)$, is that party’s degree of ‘presence’ in the system, as far as interactions with other parties are concerned.

For agents, availability is determined by the intervals during which they are reachable via the network. Some network connections (such as local area network links) may be persistent, others (such as satellite links) may be intermittently connected for fixed periods of time, while yet others (such as dial-up links) may have connection times and intervals which can only be statistically characterised.

For humans, availability is principally affected by the time spent sitting in front of their terminal (for fixed contact points) or the times they are in the vicinity of the relevant device (for mobile contact points such as cellular phones). Human AFs closely follow the natural daily cycle.

AFs are mathematically expressed as probabilities. However, AFs are not probability distributions. They rather function as look-up tables that provide a statistical measure of availability at any point in time.

AFs are assumed to be externally specified by either the relevant parties themselves or a third party such as the application programmer. Realistic AF generation is important as it directly affects the quality of the predictions, but is outside the scope of this research. Nevertheless, we have used screensaver data to generate experimental human AFs for validation purposes. Simplified versions of this data have been used to produce the figures presented in this paper.

5.2 Interaction Scheduling

Interaction scheduling is the process of co-ordinating the system's interactions such that the time of their completion is known or can be statistically predicted and guaranteed in advance. Interaction scheduling (in the TAL) is the counterpart of computation time scheduling (in the ART).

A one-way interaction (a notification) *completes* when that notification is received *and* the recipient becomes aware of it. Similarly, a two-way interaction (a request-reply) completes when the reply reaches the initiator *and* the initiator becomes aware of it. We therefore distinguish between the time a message is delivered from the time it is handled.

AFs drive not only the completion time of an interaction, but also the time of its initiation. For a guarantee level g (expressed as a probability) and a party i , an interaction can be initiated *or* completed only when $V_i(t) \geq g$.

For scheduling interactions interspersed with processing, we also distinguish between *off-line* and *on-line* processing. Off-line processing at a party can proceed independently of its availability, while on-line processing can proceed only when availability allows it, that is, when $V_i(t) \geq g$. For example, a request for a human to review a paper can proceed off-line, while requests for e-mail composition are normally done on-line.

Figure 3 shows an example of interaction scheduling for a typical request-reply scenario for $g = 0.5$. At $t = t_1$, some off-line processing has completed at j 's end and a request is ready to be sent to k . This interaction will not be initiated until $t = t_2$, when $V_j(t_2) \geq 0.5$. Party k is not aware of this until $t = t_3$, when $V_k(t_3) \geq 0.5$. As a result of the request, some on-line processing needs to be performed, proceeding only when $V_k(t) \geq 0.5$. At $t = t_4$ processing is complete. The reply can be initiated immediately, however it will not be handled by j until $t = t_5$, when $V_j(t_5) \geq 0.5$.

Interaction scheduling can be conceptually reduced to repeated applications of variations on a primitive operation mathematically formulated as:

Given a function $f(t)$, a reference point $t = t_{ref}$ and a condition C on $f(t)$, find:

$$t_x : (t_x \geq t_{ref} \wedge C(f(t_x))) \wedge \nexists t'_x : (t_x > t'_x \geq t_{ref} \wedge C(f(t'_x))) \quad (1)$$

Practically, arbitrarily-shaped functions $f(t)$ can either be stored *symbolically* in their analytic form, or *numerically* as discrete data samples. In our context, sampling has two undesirable properties:

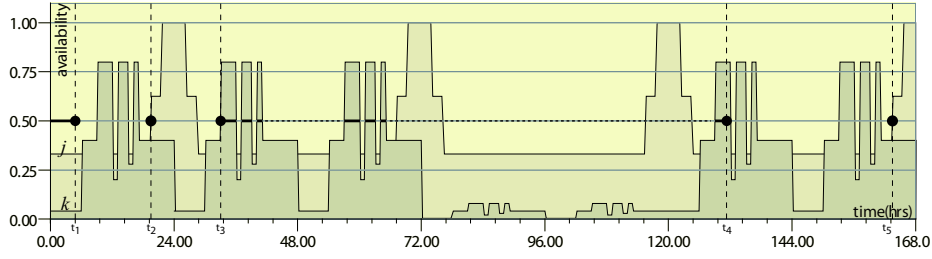


Figure 3. Interaction scheduling for a request-reply scenario between parties j and k , and a guarantee level $g = 0.5$.

- The success of sampling is dependent on a good choice of sampling frequency (step size). A large step may result in solutions landing between sampled points. A small step causes unnecessary over-sampling leading to unfeasibly large computation times. Unfortunately, in composite human-agent systems, we are dealing with interactions with greatly disparate temporal scales. On one hand, it is desirable to detect sub-second availability periods for agents, while on the other, it may be several hours or even days before human availability rises to the desired level. There is therefore no single best step size.
- Sampling is an iterative process which cannot be temporally bound without restricting the search range. This has adverse implications for any real-time system attempting to provide deterministic guarantees. A match may be found immediately or it may be, for example, that a human is on vacation and several weeks must be searched through until one is found.

To address this we have developed a stylised analytic representation for AFs (Section 5.3) which, when used in conjunction with a Selective Sampling Algorithm (Section 5.4), allows searches to be performed in bounded time.

5.3 Availability Function Representation

AF representation forms the first half of making interaction scheduling fast and predictable enough for real-time application. We constrain AF representation in the following ways:

- AFs are analytically stored and represented as hierarchical operator trees named *environments*. Primitive (leaf) nodes of environments are named *spans*. For example, a ‘working’ span for a human might set the availability to 0.9 whereas a ‘sleeping’ span might set the availability to zero.
- Spans have a *constant* availability value v .
- The periods of composed environments must be harmonics (integral multiples of one another). This is a requirement for the Selective Sampling Algorithm to work correctly. Non-harmonic environments must be broken down into a series of sub-environments such that they have harmonic repetitions.

Fortunately, most human environments follow the natural daily cycle and therefore do not require any transformation.

Environments go through a series of operations (dictated by the nodes in the tree) to arrive at the final composite AF. Nodes in the AF trees can be one of the following types:

- Activity (v): Sets the availability value to $v \in \mathbb{R}$.¹
- Bound (V, i): Bounds the value of environment V to the interval $i = [t_1, t_2]$. V remains unchanged within $[t_1, t_2]$ but becomes undefined outside it. Undefined environments do not affect the availability value.
- Repeat (V, p): Repeats environment V with a period of p seconds.
- Offset (V, o): Translates (offsets) environment V by o seconds to the right.
- Compose (f, V_1, V_2, \dots): Composes two or more environments together using the function f as an infix operator. Composition only has an effect at points where at least two environments are defined. Function f can be any binary function, but it is normally one of addition ($= V_1 + V_2$), subtraction ($= V_1 - V_2$), modulation ($= V_1 \cdot V_2$), demodulation ($= V_1 + (1 - V_1) \cdot V_2$) or selection ($= V_1$).

Environment composition for common functions f is illustrated in Figure 4. Repeated compositions result in more complex AFs, such as the ones shown in Figure 5.

Given the AF's tree representation, an evaluation function is provided to return $V(t)$ at any point t . Simplified pseudocode for evaluating the AF node at point t is given in Algorithm 1.

5.4 Selective Sampling Algorithm

The task of the Selective Sampling Algorithm (SSA) is to generate a minimal set of sampled points while guaranteeing that, if the solution of (1) exists, it will be among those points.

¹ Note that the value of v is not constrained at this stage. This allows for a simpler tree representation of AFs. However, since AFs represent probabilities, final results are clipped to lie within $[0, 1]$.

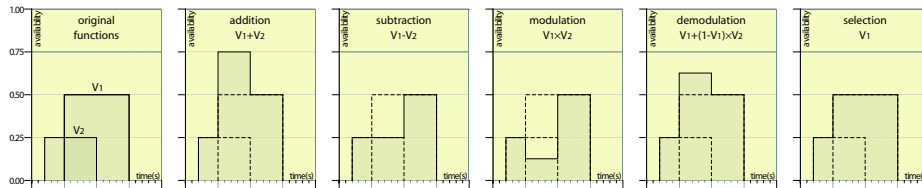


Figure 4. Results of Availability Function composition for common composition functions f (addition, subtraction, modulation, demodulation, selection).

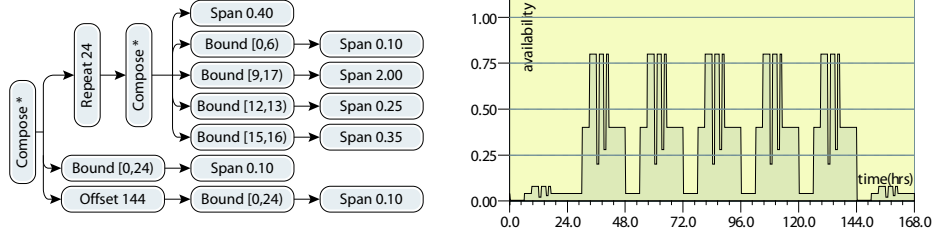


Figure 5. Tree representation and resultant Availability Function.

Informally, the SSA operates on the premise that, since spans are constant, only their (bounded) extremities are relevant. Furthermore, repeated environments generate relevant points for only their first full period – any solution found in subsequent periods will have a corresponding solution in an earlier period. Lastly, environment compositions are periodic with a period that is the maximum of the periods of their constituent environments.

The SSA can be specified as follows:

Input: An AF $V_i(t)$ and a reference point $t = t_{ref}$ at which the search starts.

Output: A set of time points where spans may cause a change in the availability value. It can be shown that this is a superset of the set of points \mathcal{S} that satisfy (1) for each discrete value that $V_i(t)$ can take, and that if a solution t_x exists, then it is always contained in \mathcal{S} .

Steps: There are two stages. In the first stage, a set of *relevant points* is generated from t_{ref} . A point is relevant if it is the earliest point where a change in spans may cause a change in the availability value. The reference point is always relevant. The theorems which drive relevant point generation for a reference point t_{ref} are summarised below (proofs are omitted):

1. Span(v) generates a single relevant point at $t = t_{ref}$.
2. Bound(V, I) generates a relevant point at each edge of I .
3. Repeat(V, p) generates relevant points only in its *relevant* period. The relevant period is the first full period nearest to t_{ref} in the direction of the search.

Algorithm 1 Availability Function evaluation algorithm.

```

evaluate(node, t)
{
  case node of type
    Span(v): v
    Bound(V,I): evaluate(V,t) if t lies in I, undefined otherwise
    Repeat(V,p): evaluate (V,remainder(t/p))
    Offset(V,o): evaluate (V,t-o)
    Compose(f,V1,V2,...): evaluate(V1,t) f evaluate (V2,t) ...
}

```

4. Offset (V, o) has a relevant point at t_x if and only if the environment V has a relevant point at $t_x - o$.
5. Compose (f, V_1, V_2, \dots) generates relevant points which always lie in the relevant periods of its constituent environments V_1, V_2, \dots .

In the second stage, each of the relevant points of the first stage functions as a reference point to add its own relevant points to the set. This stage accounts for possible availability values resulting from the composition of two or more spans.

After the second stage, the list of relevant points can be passed to the evaluation function to test (sample) each point. Figure 6 illustrates relevant point generation in the first and second stages of the SSA.

Based on this specification, correctness, completeness and complexity properties of the SSA can be proven. We omit the full format proof of these results here, however we present a sketch of the computational complexity result as it is the limiting factor for time-aware agents.

The first stage is $O(n)$ with the number of nodes in the AF representation, generating a linearly proportional number of relevant points (a maximum of two per node). The second stage is identical to the first, only t_{ref} takes values from the results of the first stage. The total algorithmic complexity is therefore $O(n^2)$. Theoretically, up to $4n^2$ points will need to be sampled in the worst case, although practically not every node will generate relevant points and many points generated from the first and second stages will overlap. The main advantage of the SSA is therefore that it completes in bounded time.

Another advantage is that, as it operates on analytic representations of AFs, it will find a solution (if one exists) irrespective how slim or far into the future it is. Agent functions that change in a temporal scale of milliseconds can be handled in the same way as human functions that may be specified in terms of hours.

Searching in ‘forward’ time can be used to predict the completion time of current interactions, while searching in ‘backward’ time can be used to determine the latest start time for an interaction sequence such that it completes by a specified deadline.

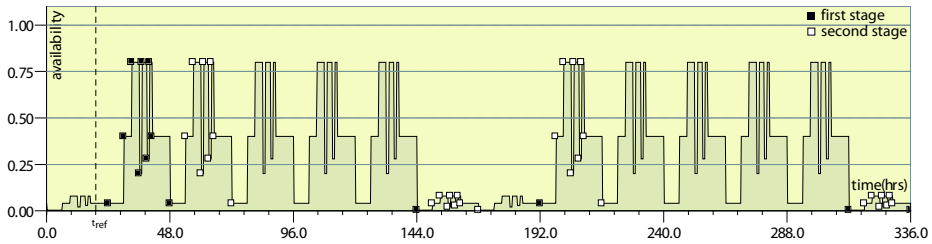


Figure 6. Relevant point generation in the first and second stages of the Selective Sampling Algorithm for a reference point at $t = t_{ref}$.

The SSA can be used as the basis for a wide range of interaction scheduling operations, including (i) finding the first point at which availability exceeds or falls below a threshold (ii) finding the maximum and minimum value of availability within an interval (iii) summing the length of time for which the availability lies above or below a threshold within an interval and (iv) calculating the value of functions of the form $h = \int_{t_1}^{t_2} (V_i(t) - g) dt$, which give a measure of how much ‘better’ or ‘worse’ availability is during $[t_1, t_2]$ with respect to g .

6 Evaluation

The SSA and the encompassing TAL layer will be evaluated from two distinct perspectives: raw real-time performance will be addressed first, followed by a brief discussion of experimental results in a practical application.

There are two versions of the SSA: one implemented in April and one in C (still accessible through April). The April version executes much slower than its C counterpart. However, its main advantage is that it is interruptible and resumable at any point by the April virtual machine. Other advantages are that Compose nodes are much more flexible, allowing any composition function (standard or user-defined) and the composition of an arbitrary number of environments in the same node.

The C version is faster than the April one by approximately 350 times. This improves the minimum temporal scale that can be handled and allows much more complex AFs without adversely affecting the system’s performance. However, once the C algorithm is started, it must execute to completion. Another limitation is that Compose nodes are restricted to pre-defined composition functions and only two environments can be composed in a single node.

Experimentally, the number of relevant points generated on average by the SSA was much lower than the worst case, lying between 1% and 2% of the theoretical $4n^2$ limit. In absolute terms, relevant point generation is very fast, with the C version generating relevant points for a 50-node AF within $160\mu s$ on a 1GHz processor. Evaluating the same function at each point takes approximately $1.2\mu s$ per point.

We are also in the process of applying this architecture in a workflow management system to schedule typical processes in Institutions of Higher Education, where some interactions are scheduled with millisecond accuracy, while the overall temporal scale can be of the order of years.

Figure 7 shows interaction scheduling for filling in a six-month progress report. Participating parties are shown on the vertical axis. Solid bars denote that a local party process is executing, dashed bars denote that the process is blocked waiting for its dependencies to be satisfied. Arrows indicate messages (interactions) between parties. Left-pointing triangles indicate the delay in interaction completion (as determined by availability), while right-pointing triangles indicate the delay in its initiation. Availability functions are plotted immediately below their corresponding party.

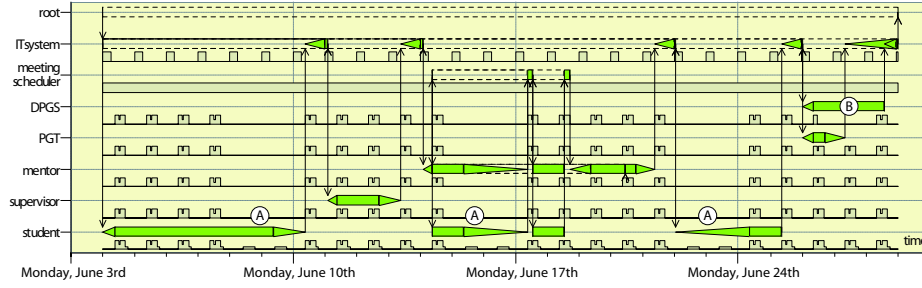


Figure 7. Interaction schedule for a “progress report” workflow process, showing (A) interactions scheduled around weekends and (B) processing time affected by periods of unavailability.

It can be seen how certain interactions occurring over weekends will not be initiated or completed until the next working day (labelled as A in the figure), as well as how on-line processing time is affected by periods of unavailability (labelled as B). This application has also demonstrated that prediction quality is directly affected by how accurately AFs reflect ‘real’ availability.

The TAL and the encompassing time-aware architecture represent ongoing research work and the current iteration has several limitations:

- The architecture is unsuited to hard real-time environments. Guarantees associated with schedules generated by the TAL are very soft and should be treated as guidelines only. This is a consequence of the presence of interactions which wholly or partly depend on humans for completion.
- In all calculations, message delivery latency is assumed to be negligible. This is a valid assumption for local controlled network environments but is not necessarily true for large-scale distributed systems over the Internet.
- Commitments to tasks may translate to changes in the AFs. For example, availability may be decreased for the duration of a task so that other tasks’ interactions are scheduled around it. These changes are currently done on a FIFO basis. This may cause a lower-priority task to be scheduled in preference to a higher-priority one. The selective re-scheduling of a minimal set of affected tasks to avoid these priority inversions has not yet been addressed.

7 Conclusions

We have presented the specification and realisation of the Time-Aware Layer in a time-aware architecture which co-ordinates agent-to-agent and human-to-agent interactions. To facilitate this co-ordination, interactions are unified by abstracting away from their temporal representation, temporal scale and class of parties they involve. This is achieved by a combination of a Unified Time Representation, common human/agent identifiers and Availability Functions.

We have used Availability Functions to model and predict the initiation and completion time of any kind of interaction. By restricting the Availability Function form with a stylised analytic representation and by taking advantage of their piecewise constant and repetitive nature, we have developed a Selective Sampling Algorithm which allows searching through them in bounded time.

The research work described in this paper is ongoing. There are still several outstanding issues, such as the effects of wide-scale distribution and the interplay between computational and interactional scheduling, which will need to be addressed in the near future. Nevertheless, initial evaluation results have been encouraging and we are applying the described time-aware architecture to implement a workflow management system.

8 Acknowledgements

This work has been undertaken with the financial support of the EPSRC-funded project TARA² (GR/N24940).

References

1. T. Babaian, B. J. Grosz, and S. M. Shieber. A Writer's Collaborative Assistant. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI2000)*, pages 7–14, 2002.
2. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, pages 35–43, May 2001.
3. V. Botti, C. Carrascosa, V. Julian, and J. Soler. The ARTIS Agent Architecture: Modelling Agents in Hard Real-Time Environments. In *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'99)*, pages 63–76. Springer-Verlag, 1999.
4. H. Chalupsky, Y. Gil, C. A. Knoblock, K. Lerman, J. Oh, D. V. Pynadath, T. A. Russ, and M. Tambe. Electric Elves: Agent Technology for Supporting Human Organizations. *Artificial Intelligence*, 23(2):11–24, 2002.
5. F. McCabe and K. Clark. April: Agent Process Interaction Language. In N. Jennings and M. Wooldridge, editors, *Intelligent Agents*, volume 890. Springer-Verlag, 1995.
6. D. Musliner, E. Durfee, and K. Shin. CIRCA: A Co-operative Intelligent Real-time Control Architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1561–1574, 1993.
7. D. Musliner, J. Hendler, A. Agrawala, E. Durfee, J. Strosnider, and C. Paul. The Challenge of Real-Time Artificial Intelligence. *Computer*, 28(1):58–66, 1995.
8. T. Payne, T. L. Lenox, S. K. Hahn, K. Sycara, and M. Lewis. Agent-Based Support for Human/Agent Teams. In *CHI 2000 Conference on Human Factors in Computing Systems*. ACM Press, 2000.
9. K. Prouskas and J. Pitt. Towards a Real-Time Architecture for Time-Aware Agents. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*, volume 1, pages 92–93, 2002.
10. T. Wagner, A. Garvey, and V. Lesser. Criteria-Directed Heuristic Task Scheduling. *International Journal of Approximate Reasoning, Special Issue on Scheduling*, 19(1-2):91–118, 1998.