# Towards a Real-Time Coordination Model for Mobile Computing

Gregory Hackmann, Christopher Gill, and Gruia-Catalin Roman

Department of Computer Science and Engineering
Washington University, St. Louis, MO, USA
{gwh2, cdgill, roman}@cse.wustl.edu

**Abstract.** Current coordination models offer limited support for applications in which mobile hosts not only must coordinate their actions, but must also coordinate when those actions will be taken. This paper describes the design of TNM, a new coordination model based on *timed futures* — a novel extension to current coordination models through which mobile hosts can propose and negotiate which actions they will take and when. We discuss the use and advantages of this new coordination model in the context of the automatic motorway application challenge problem posed for the 2005 Monterey Workshop.

## 1 Introduction

Mobile computing has experienced rapid growth in recent years due to both advances in technology and societal trends toward greater adoption of technology into peoples daily lives. These trends are occurring both on the individual scale, with mobile devices becoming the norm for personal communication, data storage and processing, and entertainment; and on larger scales as mobile network centric architectures are being designed to replace previously monolithic segments of transportation systems [1], military command and control systems [2], and other critical infrastructure.

The physical mobility of *agents*, which are people or programs performing actions that unless otherwise constrained are asynchronous and autonomous with respect to the actions of other agents in the system, is of particular importance in these systems. Physical mobility results in ad hoc networks, which create many challenges for application developers. These challenges include:

- frequent unannounced disconnections,
- message delay and loss, and
- intermittent connectivity between hosts.

To address these challenges, a number of coordination models have been developed to decouple the interactions between agents from the computations and other actions performed by each agent. This decoupling promotes rapid application development, flexible application deployment, and formal reasoning about interactions between mobile hosts. However, for some mobile computing applications (e.g., the transportation and military command and control systems mentioned above), these capabilities are not enough: the timing of actions by the mobile hosts is also critical to their correct operation, and new coordination models are needed to support temporal coordination of agents actions explicitly.

F. Kordon and J. Sztipanovits (Eds.): Monterey Workshop 2005, LNCS 4322, pp. 184–202, 2007.

This paper presents the design of TNM[1], a new coordination model that facilitates negotiation and execution of time-constrained actions across logically mobile agents and physically mobile hosts. TNM extends previous coordination models by providing new primitive tuple space operations and semantics for negotiating temporal actions, while preserving application flexibility and ease of deployment. TNM first provides future operation primitives to allow agents to propose and negotiate the future creation and removal of information from a shared tuple space. TNM then refines the semantics of those future operation primitives to include a quantitative notion of time under which explicit timing of actions can be asserted and requested. Finally, TNM defines semantics for binding proposals to requests, semantics for determining when proposals and requests for actions have been satisfied, and semantics for whether (and if so how) proposals and requests can be retracted if necessary.

The rest of this paper is structured as follows. Section 2 surveys previous related work on centralized, distributed, and timed coordination models upon which TNM builds, and which TNM extends. Section 3 describes a motivating example from the 2005 Monterey Workshop automatic motorway challenge problem, and highlights several challenges that previous coordination models do not address. Section 4 presents the TNM coordination model in detail, and identifies the main contributions of our approach: support for futures, timed futures, and semantics for satisfaction and retraction. Section 5 discusses how the TNM coordination model can address the challenges posed by the motivating example discussed in Section 3, through an example scenario. Finally, Section 6 offers concluding remarks and summarizes remaining open problems for future work.

## 2 Related Work

We now summarize related research on coordination models that precedes our work on TNM, explain how TNM builds upon and extends those previous advances, and highlight the novel aspects of our approach within that context. We first consider centralized coordination models in which agents interact through a single common tuple space that has no quantitative notion of time, in Section 2.1. In Section 2.2, we describe coordination models that add support for coordination in distributed and mobile settings. Finally, in Section 2.3 we consider coordination models that add notions of relative and absolute time. Taken together, these coordination models form the foundation upon which which we have developed our extensions in the TNM model, which we discuss in greater detail in Section 4.

### 2.1 Centralized Coordination Models

First, we consider the coordination primitives needed for centralized communication without consideration of time. The basic tuple space operations provided by LINDA [3] are:

- **out(t)**, which places *tuple* t into the tuple space;
- **in(p)**, which removes a tuple matching *pattern* p from the tuple space and returns it to the agent that issued the in operation; and

[1] TNM is an abbreviation of the Latin phrase *tempus neminem manet*, which translates into English as *time waits for no one*.

– **rd(p)**, which copies a tuple matching pattern p but does not remove the tuple from the tuple space, and returns the copy to the agent that issued the rd operation.

If no matching tuple is present when an in or rd operation is issued, the call to the operation will not return until a matching tuple is added to the tuple space. Therefore, both in and rd are *blocking* operations from the perspective of an agent invoking them.

A **react(p,s)** primitive is provided by LIME [4]. This primitive offers a natural extension to the operation primitives provided by the LINDA coordination model, in which a *reaction* containing pattern p and code fragment s is registered with the tuple space, and the call to the react operation then returns. When a tuple matching pattern p subsequently appears in the tuple space, the code fragment s is invoked. Reactions allow tuples to be handled asynchronously from the actions of the agents that register the reactions. Reactions help to protect agents from unbounded blocking.

### 2.2 Distributed and Mobile Coordination Models

Many distributed and mobile applications require bounds on blocking times and/or the ability to determine the success or failure of tuple space operations. A variety of coordination models have added variations on the in and rd primitives that are better suited to distributed coordination such as *probing* operations **inp(p)** and **rdp(p)**, which return null if no tuple matching pattern p is found; *group* blocking primitives **ing(p)** and **rdg(p)**, which return all tuples matching pattern p as opposed to only returning one matching tuple; and probing group operations **ingp(p)** and **rdgp(p)**, which return either all matching tuples or null if none are found.

LIME [4] also extends the centralized coordination model discussed in Section 2.1 by supporting transiently shared tuple spaces, to accommodate mobility of agents and hosts. LIMONE [5] adds a policy-driven acquaintance list so that agents can share tuple spaces selectively with other agents.

Again, while these extensions address many of the concerns of distributed and mobile coordination, they do not support reasoning about *whether* a particular tuple, once produced by an agent, will be visible to another agent. The TNM coordination model extends current distributed and mobile coordination models to include the ability to reason about when a proposal or request has been satisfied, as well as whether a proposal or request can be retracted if necessary.

### 2.3 Timed Coordination Models

Quantitative notions of relative and absolute time have appeared in several coordination models [6,7,8,9]. In these models, time is explicitly represented in the expression of tuples, patterns, and other coordination model features. For example, [6] defines the equivalent of an **out(t,d)** operation[2], in which the appearance of tuple t in the tuple space is subject to a relative *delay* d from when the operation is invoked.

[2] The operation signature notation we use in this paper assumes that a language that supports overloading, such as Java or C++, is used to specify a different operation definition for each distinct operation signature. In languages that do not support overloading, the same effect can be achieved through mangling of the operation names.

These timed coordination models also add further semantics to operation primitives, based on the explicit representation of time. For example, time-outs are used in the equivalent of a **rdp(p,w)** operation provided by *Timed Linda* [7] where parameter w determines how long the operation will wait before returning either a tuple matching pattern p, or null if no matching tuple appears before the time-out.

The CAST [9] coordination model provides operation primitives that take into account hosts motions in space and time. Our extensions in the TNM coordination model (described in Section 4) are based on the operations in the CAST coordination model, but only consider time and not space. To simplify discussion in the rest of this paper, we therefore abstract the CAST operations as follows:

- **out(t,start,end)**, which places *tuple* t into the tuple space — start and end are times that delimit the lifetime of tuple t in the tuple space;
- **in(p,start,end)**, which removes a tuple matching *pattern* p from the tuple space and returns it to the agent that issued the in operation — start is the time at which the operation will first attempt to match a tuple in the tuple space, and end is the time at which the operation will return null if it has not matched a tuple before then;
- **rd(p,start,end)**, which copies a tuple matching pattern p (but does not remove the tuple from the tuple space) and returns the copy to the agent that issued the rd operation — start is the time at which the operation will first attempt to match a tuple in the tuple space, and end is the time at which the operation will return null if it has not matched a tuple before then;
- **ing(p,start,end)**, which has the same semantics as in(p,start,end) except that it removes and returns all matching tuples if any are present;
- **rdg(p,start,end)**, which has the same semantics as rd(p,start,end) except that it returns copies of all matching tuples if any are present; and
- **react(p,s,start,end)**, which at time start registers a reaction containing pattern p and code fragment s with the tuple space and at time end unregisters that same reaction.

Adding quantitative time to coordination models is an important first step towards real-time coordination, as it makes it possible to reason about *when* tuples can be matched (if at all). However, these timed coordination models are still unable to reason about *whether* tuples will be matched. The TNM coordination model addresses this limitation by extending current timed coordination models to include *timed futures* that can be *bound* to other timed futures and to timed tuples, and thus can be used to reason about (and make guarantees regarding) whether tuples will be produced, read, and consumed (and if so, when).

## 3 Motivating Example

The automatic motorway application presented in the 2005 Monterey Workshop challenge problems [10] is an important example that serves to illustrate the motivation for, and potential impact of, our work on TNM. It also serves to establish criteria for determining where features provided by TNM are needed, versus where features of other existing coordination models are sufficient.

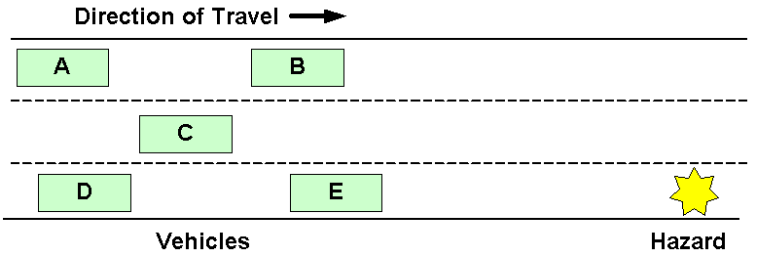


**Fig. 1.** Vehicles Approaching a Hazard

In this application, vehicles enter and leave an automatic motorway through regular entrances and exits, and navigate autonomously along the motorway.[3] To support this activity safely and with suitable performance, the vehicles' on-board computing and communication capabilities must be used to coordinate lane changes and other actions with nearby vehicles, and to propagate information about congestion, obstacles and other contextual features to other vehicles throughout the motorway.

A key problem for this system is how to ensure that traffic moves steadily throughout the system while avoiding collisions between vehicles. Consider for example a scenario illustrated in Figure 1 in which five vehicles are traveling as a *platoon* [11], occupying all three lanes of a segment of an automatic motorway. A software agent hosted by the leading vehicle in one of the lanes (vehicle E in Figure 1) detects hazards, such as a large piece of debris or other obstacle, ahead in its lane. When a hazard is detected, this agent immediately alerts software agents hosted on the other vehicles and considers actions that will allow it to avoid a collision with the hazard.

The difficulty is that an agent's appropriate actions may depend not only on current spatio-temporal factors (such as its vehicle's speed, the trajectories of the lanes in the motorway, the distance to the hazard, and the relative positions and speeds of the other cars in the platoon) but also on the potential future actions of the other agents. For example, vehicle E may be unable to move left one lane unless vehicle C decelerates, providing room for vehicle E to merge into the middle lane. However, vehicle C's deceleration may inhibit vehicle D's ability to merge left as well, to avoid the hazard. To avoid collisions (the primary concern) while preferring a smooth flow of the vehicles around the hazard (an important secondary concern), the vehicles' respective agents must therefore coordinate which actions each will take, and also at what times they will take them.

The rest of this section considers how such coordination can be realized in the context of this example scenario. In Section 3.1 we first describe relevant assumptions we make about the motorway example itself. In Section 3.2 we discuss to what extent existing coordination models address this example scenario. In Section 3.3 we then describe the remaining unaddressed issues posed by this application, which motivate the TNM coordination model that is discussed in detail in Section 4.

[3] The original application scenario proposes a two-lane motorway. We generalize this problem to have three or more lanes, in order to highlight TNM's handling of more complex coordination scenarios.

### 3.1 Assumptions About the Automatic Motorway Example

As a foundation for discussion in the rest of this paper, we now note and justify relevant assumptions we make about the automatic motorway example itself. First, we assume that accurate information about the speed and position of each vehicle can be obtained and communicated among the vehicles within a local region. Obtaining accurate positioning information is a challenging problem in its own right; but since this capability is fundamental to the safe operation of such an automatic motorway, we assume that it can be addressed through other techniques, as in [11].

Second, as stated in [10], we assume that it will not be cost-effective to deploy servers and other fixed infrastructure along the entire stretch of the motorway. For example, the cost of placing servers at motorway entrances and exits may be justified by the safety-critical nature of those junctures and by the potential for cost recovery, e.g., through collecting tolls at those servers. However, it is neither possible to anticipate all places along the motorway that a hazard could occur (e.g., where a vehicle might lose a portion of its cargo) nor would it be affordable to deploy enough servers for complete coverage of the motorway.

Third, we assume that an accurate and consistently defined view of time is available to every agent within each context: i.e., end-to-end among the servers deployed at crucial motorway intersections, within a platoon of vehicles moving along the motorway, etc. Scenarios in which previously distinct contexts merge (e.g., a platoon of cars reaches an intersection where a server is deployed) will necessarily require additional mechanisms (e.g., clock synchronization protocols) to merge the relative views of time. However, a detailed discussion of those issues is beyond the scope of this paper.

### 3.2 Applicability of Existing Coordination Models

We now consider the applicability of existing coordination models to the automatic roadway example. We first examine which features of centralized coordination models can be applied to this example. We then examine which features of distributed and mobile coordination models are required by the example, and finally which features of timed coordination models are also necessary.

*Centralized coordination models.* A server at each on-ramp and off-ramp could provide a tuple space through which agents hosted on vehicles entering and departing the motorway can exchange information through wireless communication. For example, the server at an on-ramp could provide and update a standard set of tuples describing known hazards in the motorway, current weather conditions, etc., into the tuple space. Vehicles entering the roadway could then perform rd operations to obtain that information, and as they encounter other servers along the motorway issue subsequent rd operations to refresh that information.

Agents hosted on vehicles exiting the motorway could also perform out operations to report information they have obtained while in transit. For example, an agent that has observed a hazard in the motorway could report it to the server. Each server registers reactions for various important kinds of information, such as hazard reports; when a tuple is injected into the tuple space, the associated code fragment can perform additional necessary actions (such as corroborating hazard reports, or notifying other servers along the affected segment of the motorway).

*Distributed and mobile coordination models.* In addition to the tuple space maintained on each server, we assume each vehicle would also have one or more tuple spaces (e.g., one private tuple space and one public tuple space) of its own. When a vehicle comes into wireless range of a server or another vehicle, their public tuple spaces would be federated automatically (as in LIME) so that tuples would be pooled among all connected agents. Note that for brevity we do not consider security issues, which are beyond the scope of this paper. For a sample treatment of password protection for tuples and tuple spaces and other relevant issues, we refer to our previous work on secure service provision in ad hoc networks [12].

Federating tuple spaces provides a way for vehicles to exchange information among themselves either directly or through copying tuples into the server's tuple space. For example, agents on vehicles in the roadway may supplement server-provided information with "gossip" from other vehicles. For example, uncorroborated hazard reports, which though they might be treated with lower confidence than a corroborated report issued by the server itself, would allow an agent to prepare for the possibility that a given report is in fact correct.

Furthermore, servers may themselves offer additional "quality of experience" services such as downloading information feeds for news and entertainment media from the wired network to a vehicle. Because vehicles may not know of these services *a priori*, the server could advertise these services by placing additional advertisement tuples in its tuple space. When the server's tuple space is federated with vehicles tuple spaces, those advertisements would become available to those vehicles' agents.

Other features of distributed and mobile coordination models are also needed to support the automatic motorway example. For example, the rdp operation primitive allows agents to query for tuples without blocking, which is necessary when matching tuples may or may not be present in the tuple space. Furthermore, for service discovery an agent may want to obtain the entire set of service advertisement tuples matching a service request pattern, which requires operation primitives such as rdgp.

*Timed coordination models.* The discussion so far has focused on coordination models that do not offer an explicit representation of time. However, there are many cases where the ability to represent the lifetimes of tuples, and to support timeouts and other features of operation primitives that depend on an explicit representation of time, is useful. For example, reports of current weather conditions are time sensitive, so bounding the lifetimes of tuples that report those conditions can help to avoid agents receiving out-of-date information.

Also, as vehicles move along the motorway their tuple spaces will be federated with tuple spaces on servers and other vehicles for finite intervals during which the appearance of new information may be of interest. Instead of using blocking rd or rdg operations, or transient rdp or rdgp operations, an agent on the vehicle could specify a wait time during which it would like to receive information matching a given pattern, using the timed rdp or rdgp operations.

Servers may also use timed operations to control when information is added to or removed from their tuple spaces. For example, a news feed service may be priced higher during peak hours than during off-peak hours, so that the server would specify different lifetimes for tuples advertising peak and off-peak versions of a service. The server could

also schedule the creation of service advertisement tuples by using timed out operations, which delay the tuple's creation until a specified time.

### 3.3 Remaining Challenges

Although a number of important coordination requirements for the automatic motorway can be met, as we have discussed in Section 3.2, several challenges still remain for which current coordination models do not offer adequate support. Specifically, agents need to be able to propose and request future actions, to be notified when proposals and requests are bound, to reason about when those actions will occur, and to manage retraction of proposals if subsequently available information reveals them to be infeasible or sub-optimal.

*Representing sequences of operations.* Although service discovery is supported by current coordination models, agents have no way to propose to provide a certain piece of information in the future. This constrains the scope of possible interactions significantly, as all aspects of an interaction between two agents must be handled one operation at a time, rather than allowing agents to coordinate entire sequences of operations.

For example, in the scenario presented in Figure 1, it would be at least inefficient — and due to time constraints on making decisions could possibly lead to a collision — if the vehicles had to coordinate sequentially. For example, consider the following partially ordered sequence of operations, in which the operations within each step are executed concurrently but all changes to the tuple space by those operations are completed before the next step begins:

1. Vehicle E injects a tuple alerting other vehicles to a detected hazard, and also injects a tuple with its proposal to move left one lane.
2. Vehicle D reads the alert tuple, and injects a tuple with its own proposal to move left one lane; vehicle C reads the proposal from vehicle E, determines that it must decelerate to avoid a collision with vehicle E, and injects a tuple with its proposal to decelerate.
3. Upon seeing the proposal from vehicle D, vehicle C determines that if it decelerates vehicle D may collide with it, retracts its proposal to decelerate and injects a new proposal to move left one lane.
4. Vehicle A detects that if it accelerates or maintains its velocity vehicle C may collide with it, and injects a tuple with its proposal to decelerate; vehicle B detects that if it decelerates or maintains its velocity vehicle C may collide with it, and injects a tuple with its proposal to accelerate.
5. All vehicles determine that the proposed actions will avoid collisions and the vehicles take their proposed actions.

*Timing data.* Even if agents were able to coordinate entire sequences of operations concurrently, the coordination model still would not provide a means for them to reason about whether a given sequence of operations would complete within a specified time limit (e.g., before vehicle E came too close to the hazard to be able to steer around it). To perform this kind of reasoning, agents must be able to propose not only sequences

of actions to be taken, but the times at which each of the actions in a sequence would be taken. Therefore, the coordination model needs to incorporate an explicit quantitative representation of time into its support for coordinating sequences of actions.

*Proposals, requests, and binding.* While it is necessary for vehicles to add timing *data* to the tuples they inject, this alone is not sufficient to allow efficient and reliable coordination. For example, if during step 1 vehicle A had decided to change lanes (e.g., to be able to exit the roadway at an upcoming exit) and during step 2 read vehicle C's proposal to decelerate, then vehicle C's retraction of its proposal to decelerate would violate vehicle A's dependence on that proposal and furthermore vehicle C would have no indication of that violation. What is needed, therefore, is a way for timed tuples to be both *proposed* and *requested*, and for proposals and requests to be *bound* so that reliable coordination involving inter-dependent sequences of operations can be achieved.

*Retraction.* Anticipating and addressing all combinations of events that may have significant consequences is notoriously difficult in complex inter-connected applications, even within a single mobile host, as the Mars Pathfinder [13] example demonstrates. Should an unanticipated condition arise on a host during coordination involving sequences of actions, an agent may need to retract a proposed sequence of actions. However, the implications of such a retraction must be handled carefully since other agents may have made their own decisions based on the assumption that the proposed sequence of operations would occur. Two distinct forms of retraction are needed:

- *strong retractions* in which the proposed sequence of operations is retracted unconditionally (e.g., if a vehicle enters a mode of failure after which it is impossible for it to perform an expected action); and
- *weak retractions* in which the proposed sequence of operations is retracted only if another agent does not depend on the sequence already (e.g., if a vehicle discovers a potential optimization to an already sufficient maneuver that it has proposed).

Although the addition of timing data to tuples has been achieved in other coordination models as we have discussed in Section 2, explicit mechanisms for proposing, requesting, binding, and retracting future operations are still needed. In Section 4 we discuss our proposed approach to address these outstanding issues.

## 4 The TNM Coordination Model

This section describes the main contributions of this work, which address the unmet challenges described in Section 3.3: (1) extending coordination models with *future* primitives to let agents propose and request the future availability of information within a shared tuple space, which we discuss in Section 4.1; (2) extending untimed futures to include a quantitative notion of time, which we discuss in Section 4.2, so that explicit real-time constraints can be asserted, evaluated, and assured; and (3) extending the semantics of untimed and timed futures to address when futures are *bound* and when futures may or may not be *retracted*, which we discuss in Sections 4.3 and 4.4, respectively.

### 4.1 Futures

We first provide a new extension to existing coordination models: "future" primitives that propose and request basic tuple-space operations which would take place in the future. Five future primitives are provided by the TNM coordination model, which agents can use to propose to inject, remove, or copy tuples into or from the tuple space:

- **future_out(p)** proposes to put a tuple matching pattern p into the tuple space at some point in the future;
- **future_in(p)** requests to remove a tuple matching pattern p from the tuple space at some point in the future;
- **future_rd(p)** requests to copy a tuple matching pattern p from the tuple space at some point in the future;
- **future_ing(p)** requests to remove all tuples matching pattern p from the tuple space at some point in the future; and
- **future_rdg(p)** requests to copy all tuples matching pattern p from the tuple space at some point in the future.

Each of these operations puts a special *future* tuple of the form

```
<future_type, id, pattern, bound_list>
```

into the tuple space. The future_type field indicates what type of future it is (i.e., future_out, future_in, future_rd, future_ing, or future_rdg). The id field contains a unique identifier that is assigned automatically to that tuple by the tuple space. The pattern field contains a copy of the pattern that was passed as a parameter to the future operation. The bound_list field is initially empty and is updated automatically according to the binding rules discussed in Section 4.3. After the tuple is placed in the tuple space and its id and bound_list fields are filled in, each of these future operations then returns a copy of the newly created and initialized future tuple. The agent may then use that tuple's identifier at a later time to request retraction of that future tuple, as we discuss in Section 4.4.

### 4.2 Timed Futures

As we discussed in Section 3.3, not only must a real-time coordination model support agents' ability to reason about sequences of actions, but it must also allow them to reason about *when* those actions would occur. To provide this capability, the TNM coordination model includes explicit representations of time in the future operations, and adds temporal semantics to the binding and preference rules for timed futures.

We first introduce timed versions of the future_in, future_rd, and future_out primitives. These primitives are similar to the corresponding untimed future operation primitives, except that they specify temporal parameters in addition to a pattern:

- **timed_future_out(p,start,end)**, proposes that a timed out operation will put a timed tuple matching pattern p into the tuple space — the lifetime of the tuple will be delimited by the times given by the start and end parameters to the timed future out operation;

- **timed_future_in(p,start,end)**, requests a timed in operation that will attempt to remove a tuple matching pattern p from the tuple space at the time given by the start parameter, and will wait until the time given by the end parameter before returning null if no tuple is matched by then;
- **timed_future_rd(p,start,end)**, requests that a timed rd operation will attempt to copy a tuple matching pattern p from the tuple space at the time given by the start parameter, and wait until the time given by the end parameter before returning null if no tuple is matched by then;
- **timed_future_ing(p,start,end)**, requests a timed ing operation that will attempt to remove all tuples matching pattern p from the tuple space at the time given by the start parameter, and will wait until the time given by the end parameter before returning null if no tuple is matched by then; and
- **timed_future_rdg(p,start,end)**, requests that a timed rdg operation will attempt to copy all tuples matching pattern p from the tuple space at the time given by the start parameter, and will wait until the time given by the end parameter before returning null if no tuple is matched by then.

Each of these operations puts a special *future* tuple of the form

```
<future_type, start, end, id, pattern, bound_list>
```

into the tuple space. The future_type field again indicates what type of future it is (i.e., timed_future_out, timed_future_in, timed_future_rd, timed_future_ing, or timed_future_rdg). The start and end fields contain the values of the temporal parameters passed to the timed future operation primitive. The id, pattern, and bound_list fields are equivalent to the corresponding fields in the untimed future tuples, and uniqueness of id field values is maintained across both timed and untimed future tuples.

### 4.3 Binding

A novel and important feature of the TNM coordination model is its support for binding future tuples so that dependences of requested tuples on proposed tuples can be represented and reasoned about. We first consider the semantics of binding untimed futures, and then discuss the semantics of binding timed futures. For both untimed and timed futures, we present rules for which tuples can be bound and then present preference rules which govern the order in which tuple bindings are performed among those that can be bound.

*Semantics of binding.* We say that a future_in, future_ing, future_rd or future_rdg tuple with a pattern $p$, and a future_out tuple with a pattern $p'$, can be *bound* automatically by the tuple space if $p$ is no more specific than $p'$, i.e., $\{t \mid t \,\text{matches}\, p'\} \subseteq \{t \mid t \,\text{matches}\, p\}$. A bound future_in, future_ing, future_rd or future_rdg tuple with pattern p constitutes a guarantee that a subsequent in, ing, rd, or rdg operation (respectively) with pattern parameter q will succeed as long as pattern q is no more specific than pattern p. In fact, the operation can succeed if any future_out operation produces a tuple that matches pattern q. However, it is impossible to *reason* about such a guarantee unless q is no more specific than p.

Timed future tuples can be used to determine if there will be a timed out operation that is guaranteed to satisfy a timed in, ing, rd, or rdg operation, and to bind timed future tuples in a manner similar to the binding of untimed future tuples. We say that a timed_future_in tuple, a timed_future_ing tuple, a timed_future_rd tuple, or timed_future_rdg tuple, with a pattern $p$ and start time $s$ and end time $e$, and a timed_future_out tuple with a pattern $p'$ and start time $s'$ and end time $e'$, can be *bound* automatically by the tuple space if $p$ is no more specific than $p'$, and the time interval delimited by $s$ and $e$ overlaps the time interval delimited by $s'$ and $e'$. We define the *temporal intersection* of two timed future tuples to be the intersection of the time intervals delimited by their respective start and end fields.

*Binding rules.* Binding of untimed future tuples is a directed relation from future_out tuples to future_in, future_rd, future_ing, and future_rdg tuples. The list in the bound_list field of a future_out tuple may only contain identifiers of other kinds of future tuples, and the list in the bound_list field of all other kinds of future tuples may only contain identifiers of future_out tuples. Because it is possible for multiple future tuples to have patterns matching the pattern in a newly injected future tuple, the tuple space respects the following rules when binding future tuples, to preserve appropriate semantics of the future operations.

- A future_in or future_rd tuple may be bound by exactly one future_out tuple.
- A future_ing or future_rdg tuple may be bound by any number of future_out tuples.
- A future_out tuple that is bound to a future_in or future_ing tuple may not be bound to any other future tuple.
- A future_out tuple may be bound to any number of future_rd or future_rdg tuples.

The binding rules for timed future tuples are similar to those for untimed future tuples, but with the addition of temporal semantics. With timed future tuples binding is a directed relation from timed_future_out tuples to timed_future_in, timed_future_rd, timed_future_ing, and timed_future_rdg tuples.

- A timed_future_in or timed_future_rd tuple may be bound by exactly one timed_future_out tuple.
- A timed_future_ing or timed_future_rdg tuple may be bound by any number of timed_future_out tuples.
- A timed_future_out tuple that is bound to a timed_future_in or timed_future_ing tuple may not be bound to another timed_future_in or timed_future_ing tuple.
- A timed_future_out tuple can only be bound to future_rd or future_rdg tuples whose end times are earlier than the start time of any timed_future_in or timed_future_ing tuple to which the timed_future_out tuple is bound, but the timed_future_out tuple can be bound to any number of such future_rd or future_rdg tuples.

*Preference Rules.* When a new future tuple is injected into the tuple space, the tuple space will perform a sequence of tuple bindings made one-at-a-time according to the rules given above, until no more allowed bindings can be made. At each step of that sequence, the following preference rules are applied (in the order they are given) to the set of all allowed bindings, to choose the next binding to be made by the tuple space.

- Preference is first given for binding a future_out tuple to a future_in, future_rd, future_ing or future_rdg whose pattern is no less specific to its pattern than any other future_in, future_rd, future_ing or future_rdg tuple. This preference rule is designed to increase specificity of the bindings performed.
- Preference is then given for binding a future_out tuple to an unbound future_in, future_rd, future_ing or future_rdg tuple over binding it to an already bound future_ing or future_rdg tuple. This preference rule is designed to decrease the number of unbound requests for tuples.
- Preference is then given for binding a future_out tuple to an unbound future_rd or future_rdg tuple over binding it to an unbound future_in or future_ing tuple. This preference rule is designed to increase the number of requests that can be satisfied by each proposal by binding it preferentially to requests that leave the tuple available to other requests rather than consuming it.
- If more than one binding has been selected after the previous preference rules have been applied, then one binding is chosen non-deterministically from that selected set. This preference rule is designed to ensure that binding progresses even when the preceding preferences have not distinguished a unique binding.

Preference rules for timed future operation primitives are similar to those for untimed future operation primitives, though both pattern specificity and temporal specificity are considered.

- The first preference is for binding a timed_future_out tuple to a timed_future-_in, timed_future_rd, timed_future_ing or timed_future_rdg whose pattern is no less specific to its pattern than any other matching timed_future_in, timed-_future_rd, timed_future_ing or timed_future_rdg tuple. Like the corresponding preference rule for untimed futures, this preference rule is designed to increase specificity of the bindings performed in terms of the tuples' *patterns*.
- The next preference is to bind a timed_future_out tuple to a timed_future_in, timed_future_rd, timed_future_ing or timed_future_rdg whose temporal intersection no smaller than that for any other matching timed_future_in, timed_-future_rd, timed_future_ing or timed_future_rdg tuple. This preference rule is designed to increase specificity of the bindings performed in terms of the tuples' *temporal intervals*.
- Preference is then given for binding a timed_future_out tuple to an unbound timed_future_in, timed_future_rd, timed_future_ing or timed_future_rdg tuple over binding it to an already bound timed_future_ing or timed_future_rdg tuple. Like the corresponding preference rule for untimed futures, this preference rule is designed to decrease the number of unbound requests for tuples.
- Preference is then given for binding a timed_future_out tuple to an unbound timed_future_rd or timed_future_rdg tuple over binding it to an unbound timed_future_in or timed_future_ing tuple. Like the corresponding preference rule for untimed futures, this preference rule is designed to increase the number of requests that can be satisfied by each proposal by binding it preferentially to requests that leave the tuple available to other requests rather than consuming it.
- If more than one binding has been selected after the previous preference rules have been applied, then one binding is chosen non-deterministically from that selected

set. Like the corresponding preference rule for untimed futures, this preference rule is designed to ensure that binding progresses even when the preceding preferences have not distinguished a unique binding.

### 4.4 Satisfaction and Retraction

Because each future tuple represents a proposed future operation on the tuple space, two additional concerns must be addressed: when is a future considered to have been satisfied by another operation on the tuple space, and when may a future tuple that has not been satisfied be retracted from the tuple space by the agent that proposed it? We augment the structure of timed and untimed future tuples to contain two additional boolean fields labeled *satisfied* and *retracted*, which are both valued false when the future tuple is created. The satisfied field indicates whether or not a future tuple has been satisfied by a subsequent basic tuple space operation, and the retracted field indicates whether or not a future tuple has been retracted by an agent.

*Satisfaction.* Each basic tuple space operation can satisfy only one previously unsatisfied future tuple corresponding to that operation type: an out operation can only satisfy a future_out or timed_future_out tuple, an in operation can only satisfy a future_in or timed_future_in tuple, and so on. Furthermore, a basic tuple space operation can only satisfy a future tuple whose pattern is no more specific than the pattern or tuple it was given, and can only satisfy a timed future tuple whose start and end times delimit an interval within which the basic tuple space operation was performed.

If multiple future tuples could be satisfied by a basic tuple space operation, the following rules are applied (in the order they are given) to determine which future tuple is marked as being satisfied, taking into account the pattern and temporal specificity of the basic tuple space operation to the future tuples that it could potentially satisfy.

- Preference is given first to future tuples whose pattern is most specific to the pattern or tuple given to the basic tuple space operation.
- Preference is then given to timed future tuples over untimed future tuples and to timed future tuples with an earlier end time over timed tuples with a later end time.
- If more than one future tuple has been selected after the previous satisfaction rules have been applied, then one future tuple is chosen non-deterministically from that selected set.

The future tuple thus selected is then updated automatically by the tuple space by setting the tuple's satisfied field to true.

*Retraction.* The final issue we address in the TNM coordination model is the ability for agents to retract future tuples. This ability supports re-negotiation of (timed or untimed) sequences of actions among agents, to allow sequences that are no longer feasible to be discarded, or to allow more optimal sequences to be chosen, as the agents' operating contexts change, or as agents obtain more accurate information about their operating contexts, over time.

However, the ability of an agent to retract a proposed operation has important implications for other agents whose actions depend on the operation that would be retracted.

For example, retracting a future_out tuple takes back the proposal it has made that a suitable tuple would be provided by a subsequent out operation, upon which another operation requested by a future_in, future_ing, future_rd or future_rdg operation may depend. This can have cascading effects on agents expecting the no-longer-guaranteed tuples to be available which could then prevent them from providing tuples they had proposed. Despite the potentially adverse consequences of retraction, possibilities such as the anticipated failure of a host[4] motivate having operation primitives for both strong and weak forms of retraction in the TNM coordination model:

- **retract(id)**, where the future tuple whose unique id field value matches the passed id parameter is simply marked as having been retracted, if it is present in the tuple space; and
- **retractp(id)**, where the matching future tuple is only marked as having been retracted if that will not affect other agents, i.e., if it has already been satisfied or its bound_list field is empty.

The **retract(id)** operation returns a copy of the future tuple that was marked as having been retracted, or null if no tuple with the given id was present in the tuple space. The **retractp(id)** operation returns a copy of the future tuple that was marked as having been retracted, or null if no future tuple with the given id was present in the tuple space or if the tuple was bound to any other tuples.

To ensure that appropriate semantics for the future tuples can be enforced entirely within the future and retraction operation primitives, we place one restriction on the out, in, and ing tuple space operations of the TNM coordination model: they can be applied to any kind of tuple *except* the timed and untimed future tuples described in Sections 4.1 and 4.2. An out operation that is given a tuple whose first field contains a future tuple type will simply return without modifying the tuple space. An in or ing operation with a pattern that can match a future tuple will ignore all future tuples. The rd and rdg operations have the same semantics for future tuples as they would to any other tuples, however, so that an agent may query for copies of future tuples existing in the tuple space.

## 5 Discussion

A crucial feature of the TNM coordination model is that agents can propose and request when tuples will be produced, read, and consumed, which can be mapped directly onto spatio-temporal properties of the context within which agents will propose and take their actions. For example, in the hazard avoidance scenario shown in Figure 1 and discussed in Section 3, agents running on each vehicle could propose timed future out operations and request timed future rd operations for tuples describing timed sequences of acceleration, deceleration, and lane change actions. In the following discussion we assume that agents exchange information about the vehicles' motions frequently, so that

[4] For the sake of discussion we assume hosts can detect and announce their failure prior to its occurrence - this assumption can be weakened in practice through periodic heartbeats, failure detectors, and other means.

each maintains an accurate picture of the spatio-temporal structure of the platoon, and can compute potential actions accordingly.

When the agents on the vehicles in the platoon receive the hazard alert from the agent on vehicle E, each generates plausible sequences of actions that it could take to make room for other vehicles, or to avoid the hazard itself. The crux of negotiating a plausible maneuver that will avoid collisions is the position of vehicle C among the other vehicles, and the inter-dependence of vehicles actions that this entails. For example, one plausible sequence of actions would be for vehicle C to accelerate and for vehicles D and E to decelerate and then change lanes, with the resulting configuration of the platoon illustrated in Figure 2. Another plausible sequence of actions would be for vehicle C to maintain its current trajectory, for vehicle E to accelerate and change one lane left ahead of vehicle C, and for vehicle D to decelerate and change one lane left behind vehicle C, with the resulting configuration of the platoon shown in Figure 3. A third plausible sequence of actions would be for vehicle A to decelerate and for vehicle B to accelerate, and for vehicles C, D, and E to change one lane left, resulting in the configuration of the platoon shown in Figure 4. Figure 5 summarizes the set of possible maneuvers by each of the vehicles, combinations of which produce the platoon configurations illustrated in Figures 2 through 4:

1. vehicle A would decelerate until time $t_1$;
2. vehicle B would accelerate until time $t_1$;
3. vehicle C would decelerate until time $t_2 > t_1$;
4. vehicle C would maintain velocity until time $t_1$ and then change one lane left;
5. vehicle C would accelerate until time $t_2 > t_1$;
6. vehicle D would decelerate until time $t_2$ and then change one lane left;

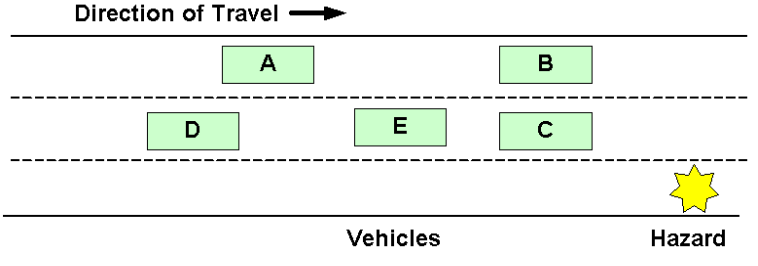


**Fig. 2.** Vehicle C Accelerates

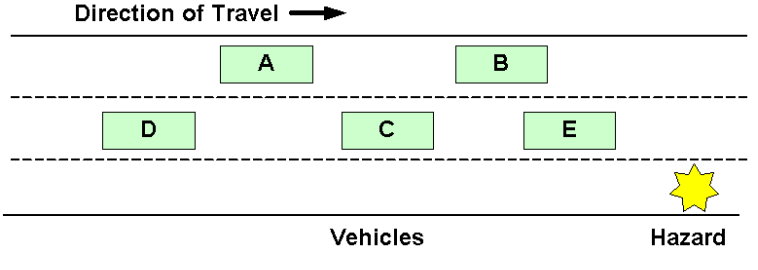


**Fig. 3.** Vehicle C Maintains Trajectory

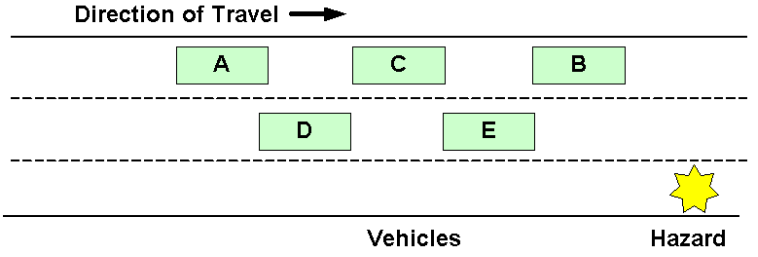


**Fig. 4.** Vehicle C Moves Left

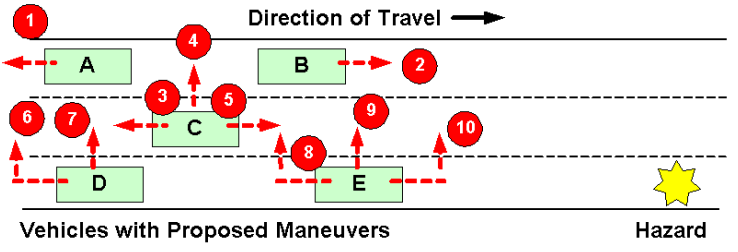


**Fig. 5.** Proposed Vehicle Maneuvers

7. vehicle D would maintain velocity until time $t_2$ and then change one lane left;
8. vehicle E would decelerate until time $t_2 > t_1$ and then change one lane left;
9. vehicle E would maintain velocity until time $t_2 > t_1$ and then change one lane left; and
10. vehicle E would accelerate until time $t_2 > t_1$ then change one lane left.

Each vehicle's agent would propose its own actions, and would also request actions by agents on nearby vehicles. For simplicity we assume that each of the agents can compute both its own plausible actions and those of the agents on other vehicles[5]:

- vehicle A would use timed_future_rd operations to request action 2 of vehicle B (i), and actions 3 through 5 of vehicle C (ii); vehicle A would then propose action 1 by using a timed_future_out operation (iii);
- vehicle B would use timed_future_rd operations to request action 1 of vehicle A (iv, which would be bound by vehicle A's timed_future_out operation from iii), and actions 3 through 5 of vehicle C (v); vehicle B would then propose action 2 by using a timed_future_out operation (vi, which would bind to vehicle A's timed_future_rd from i);
- vehicle C would use timed_future_rd operations to request action 1 of vehicle A (vii, which would be bound by vehicle A's timed_future_out operation from iii), action 2 of vehicle B (viii, which would be bound by vehicle B's timed_future_out operation

[5] This assumption could be relaxed in practice by having agents register reactions for other agents requests, determine whether or not they can perform those requested actions, and if they are plausible propose them as well.

from vi), actions 6 and 7 of vehicle D (ix), and actions 8 through 10 vehicle E (x); vehicle C would use timed_future_out operations to propose actions 3 through 5 (xi, which would bind to vehicle A's and vehicle B's timed_future_rd operations from ii and v);
- vehicle D would use timed_future_rd operations to request actions 3 through 5 of vehicle C (xii, which would be bound by vehicle C's timed_future_out operations from xi) and actions 8 through 10 of vehicle E (xiii); vehicle D would use timed_future_out operations to propose actions 6 and 7 (xiv, which would bind to vehicle C's timed_future_rd operations from ix);
- vehicle E would use timed_future_rd operations to request actions 3 through 5 of vehicle C (xv, which would be bound by vehicle C's timed_future_out operations from xi) and actions 6 and 7 of vehicle D (xvi, which would be bound by vehicle D's timed_future_out operations from xiv); vehicle E would use timed_future_out operations to propose actions 8 through 10 (xvii, which would bind to vehicle C's and vehicle D's timed_future_rd operations from x and xiii).

Each vehicle would issue reactions for bound timed future tuples corresponding to the actions it has proposed and requested. When an agent's proposal or request is bound, the agent can use weak retraction to remove other proposed or requested actions from consideration, though agents may wait until at least one complete plausible sequence of inter-dependent actions by all vehicles is decided before pruning the set of potential sequences by retracting requests or proposals.

## 6 Concluding Remarks

In this paper we have presented the TNM coordination model, which offers novel coordination features including futures, timed futures, binding, and satisfaction and retraction semantics, which allow agents to negotiate timed and untimed sequences of actions explicitly. We have shown how these features can help to address challenges posed by the 2005 Monterey Workshop automatic motorway example, which other coordination models do not address. As our experience with avionics mission computing applications we have studied in previous research [14, 15, 16] indicates, automating vehicle functions (including real-time coordination among moving vehicles) to increase each agent's situational awareness and responsiveness is likely to be beneficial in practice.

As future work we will examine how the tuple space can automatically rebind future tuples to preserve guarantees that would otherwise be disrupted by strong retraction semantics, and will study whether this automatic rebinding capability could allow weak retraction to be realized under a wider set of circumstances, e.g., for automatic optimization under control of the tuple space as new future tuples are proposed and retracted. We will also investigate the complex scheduling issues which arise from placing timing constraints on tuple-space operations. Many metrics that are used in real-time scheduling, such as per-process priorities, have limited applicability in ad-hoc networks of mobile hosts. Furthermore, centralized scheduling algorithms would be impractical to use in mobile ad-hoc networks, as hosts move into and out of contact with each other. Therefore, we plan to investigate new scheduling algorithms for timed future coordination, to provide resource-limited timing guarantees in a decentralized fashion.

## References


1. Caltrans, the Institute of Transportation Studies at UC Berkely: California partners for advanced transit and highways (path). (http://www-path.eecs.berkeley.edu)
2. Army, U.S.: Future combat systems (fcs). (http://www.army.mil/fcs/)
3. Gelernter, D.: Generative communication in Linda. ACM Trans. Program. Lang. Syst. **7**(1) (1985) 80–112
4. Picco, G., Murphy, A., Roman, G.C.: LIME: Linda meets mobility. In: Proc. of the $21^{st}$ Int'l. Conf. on Software Engineering. (1999)
5. Fok, C.L., Roman, G.C., Hackmann, G.: A lightweight coordination middleware for mobile computing. In: 6th International Conference on Coordination Models and Languages (Coordination '04). (2004) 135–151
6. Jacquet, J.M., Bosschere, K.D., Brogi, A.: On timed coordination languages. In: 4th International Conference on Coordination Languages and Models (Coordination '00), London, UK, Springer-Verlag (2000) 81–98
7. de Boer, F.S., Gabbrielli, M., Meo, M.C.: A denotational semantics for timed linda. In: 3rd ACM SIGPLAN international conference on Principles and practice of declarative programming (PPDP '01), New York, NY, USA, ACM Press (2001) 28–36
8. Mousavi, M.R., Reniers, M., Basten, T., Chaudron, M.: Separation of concerns in the formal design of real-time shared data-space systems. In: Third International Conference on Application of Concurrency to System Design (ACSD '03), IEEE Computer Society (2003)
9. Roman, G.C., Handorean, R., Sen, R.: Tuple space coordination across space and time. In: 8th International Conference on Coordination Languages and Models (Coordination '06), Bologna, Italy, Springer LNCS 4038 (2006) 266–280
10. Kordon, F., Sztipanovits, J.: The monterey workshop series (2005 theme: Workshop on networked systems: realization of reliable systems on top of unreliable networked platforms). http://www-src.lip6.fr/homepages/Fabrice.Kordon/Monterey/objectives.html (2005)
11. Horowitz, R., Varaiya, P.: Control design of an automated highway system. Proceedings of the IEEE **88**(7) (2000) 913–925
12. Handorean, R., Roman, G.C.: Secure Service Provision in Ad Hoc Networks. In: Proceedings of the First International Conference on Service Oriented Computing (ICSOC 2003). (2003)
13. Jones, M.: What really happened on Mars? www.research.microsoft.com/∼mbj/Mars_Pathfinder/Mars_Pathfinder.html (1997)
14. Doerr, B.S., Venturella, T., Jha, R., Gill, C.D., Schmidt, D.C.: Adaptive Scheduling for Real-time, Embedded Information Systems. In: Proceedings of the 18th IEEE/AIAA Digital Avionics Systems Conference (DASC). (1999)
15. Gill, C., Schmidt, D.C., Cytron, R.: Multi-Paradigm Scheduling for Distributed Real-time Embedded Computing. IEEE Proceedings, Special Issue on Modeling and Design of Embedded Software **91**(1) (2003)
16. Gill, C.D., Gossett, J.M., Corman, D., Loyall, J.P., Schantz, R.E., Atighetchi, M., Schmidt, D.C.: Integrated Adaptive QoS Management in Middleware: An Empirical Case Study. Journal of Real-time Systems **24** (2005)