Coordinating Asynchronous and Open Distributed Systems under Semiring-Based Timing Constraints

Yue Yu, Shangping Ren[1]

Illinois Institute of Technology
Chicago, IL 60616, USA

Carolyn Talcott[2]

SRI International
Menlo Park, CA 94025, USA

Abstract

For asynchronous and open distributed systems, dynamicity, openness, and stringent quality of service requirements post great challenges to model and develop such systems. The Actor-Role-Coordinator (ARC) model was previously proposed to address these challenges. The role concept in the model attends to the dynamicity and openness issues by providing abstractions of actor behaviors. In this paper, we focus on coordinating actors and roles through message manipulations based on synchronous event-based timing constraints. In addition, different types of timing constraints are generalized into a semiring-based constraint structure; and the all-pairs extremal paths algorithm on closed semirings is applied to derive the most stringent constraints which are logical implications of the original set of constraints. The derived implicit constraints are further used to test constraint inclusions and decide intersections between feasible regions of timing constraint sets. The integration of the ARC model and the semiring-based timing constraint models is prototyped through Maude, a rewriting logic language. We further use the approach to solve the Restaurant for Dining Philosophers problem and illustrate the expressiveness of the ARC and the semiring-based timing constraint models for exogenous and composable coordination of open systems.

Keywords: Coordination model, timing constraint model, ARC, Maude

1 Introduction

The proliferation of embedded devices and significant advances of wireless network technologies have led to new applications that involve an increasingly *large* number of *dynamically changing* systems of objects interacting *asynchorously*. These objects oftentimes must together satisfy multiple types of *Quality-of-Service* (QoS) requirements. As such, the need for a new paradigm to reduce the complexity and ease the development of these applications is growing.

Viewing asynchronous and open distributed applications as compositions of coordination and concurrent computation decouples the two concerns and allows higher levels of

[1] Email: {yyu8,ren}@iit.edu
[2] Email: clt@cs.stanford.edu

abstraction. However, these advantages can only be fully realized if the following two fundamental requirements are met. First, it is essential to have a coordination model that focuses on coordination under constraints, and is decentralized, exogenous, scalable, and able to handle dynamic concurrent computation without itself being dynamic. Second, in order to reason about constraints, a formal model that can uniformly represent these different types of constraints must be provided.

Our earlier research on coordination models has resulted in a layered coordination model, the Actor-Role-Coordinator (ARC) model [26], which specifically targets for the first need. This paper is to address the second requirement listed above.

1.1 Related Work

Coordination is an important paradigm for asynchronous and open distributed applications. A wide spectrum of coordination strategies have been proposed to capture the functional aspects of these applications. In the landmark survey [24], Papadopoulos et al. conclude that coordination models can be classified into two categories, data-driven and control-driven. Linda [16] and its mobile extension, Lime [25], KLAIM [13] and its stochastic extension [14] represent the data-driven category; while the IWIM or Manifold [2] presents a control-driven or "exogenous" category. Tuple center [21] and ReSpecT [20] provide a hybrid view.

Control-driven models isolate coordination by considering functional entities as black boxes. For example, the Abstract Behavior Type (ABT) model [4] and its language Reo [3] extend the IWIM by treating both computation and coordination components as composable ABTs. The emphasis in Reo is on the connectors, and the coordination and communication patterns which they impose on the components, but not on the components which are the entities being coordinated (coordinatees). Moreover, specifications of timing constraints are supported in the Timed Data Stream (TDS) semantics of Reo. Some control-driven models, such as TuCSoN with ACC [22], CoLaS [11], and ROAD [10], address the scalability issues of open distributed systems through the concept of groups.

The ARC model [26] partitions coordination into two disjoint categories, i.e., *intra-role* and *inter-role* coordination, and uses roles and coordinators, respectively, to abstract these behaviors (see Section 2). The coordinatees in the ARC model are actors [1] which are computational entities that interact by asynchronous message exchange. Coordination in ARC is achieved through exogenous message manipulations in *space* and *time* (constraining message destination and dispatch time) which are transparent to the coordinated actors. Our earlier paper [28] gives detailed comparison of the ARC model with the Reo, the Reflect Russian Dolls (RRD) [18], and other coordination models.

While spatial manipulations of messages are carried out by roles rerouting messages to destinations based on role policies [26], a formal model to specify and verify temporal manipulations of messages is presented in this paper.

Incorporating the notion of time into coordination models is not new. Papadopoulos [23] combines IWIM with the work on timed concurrent constraint programming [27]; and several extensions of Linda with different notions of time are introduced in [17]. This paper differs from previous work in that we provide a higher abstraction of timing requirement that does not depend on any specific type of timing constraints. Semirings have been proposed as a framework for generalizing, composing, and relating quality of service constraints [6,12]; and it has been shown to be applicable to component-based models [7,29] where weights on connectors (representing QoS constraints) as well as their compositions are modeled by semirings. Therefore, it is quite natural to abstract different types of timing constraints through semirings. In this paper, we use constraint semirings to generalize timing constraints; and more importantly, we study the properties of the feasible region allowed by a set of semiring-based timing constraints. Algorithms for solving extremal path problems in directed graphs based on closed semirings [15] allow us to derive implicit timing constraints in a general form. Such implicit constraints are crucial in comparing the feasible regions of semiring-based timing constraints.

1.2 Main Contributions

The main contributions of this paper are twofold. Firstly, coordination constraints are mapped into semiring-based timing constraints and their effects on actor computations are studied. Linear programming duality of the shortest path problem allows us to give necessary and sufficient conditions for the inclusion relation between feasible regions of timing constraint sets in the real-time case. Formal proofs of the main properties (Lemma 3.2 and Theorem 3.3) are given in the appendices. The result is further generalized to semiring-based timing constraints based on morphisms between semirings. Secondly, the ARC model and the semiring-based timing constraints are integrated through the Maude [9] specification language for exogenous and composable coordination of open systems. We use a canonical open distributed system example, the Restaurant for Dining Philosophers [8], to illustrate such integration.

1.3 Road Map

The rest of this paper is organized as follows: for self-containment, Section 2 gives a brief description of the ARC model. A detailed description of the ARC model can be found in [26,28]. Section 3 discusses semiring-based timing constraints and their properties. Section 4 presents a specification of the ARC model and semiring-based timing constraints in Maude and gives an example to show how such a formal specification facilitates reasoning about coordination properties. Finally, we conclude in Section 5.

2 The Actor-Role-Coordinator Model

The Actor-Role-Coordinator (ARC) model [26,28] is a role-based coordination model in which a role is a static abstraction for a set of behaviors that the underlying actors share. The actor model [1] is used to model the distributed system's underlying computation. The functionality of the role is to coordinate its members. This type of coordination is called intra-role coordination. The intra-role coordination is achieved through policy-based message rerouting and reordering among actors within the same role. Coordination among different roles, i.e., inter-role coordination, on the other hand, is done by coordinators. Coordinators constrain roles' coordination behaviors which eventually affects message dispatch time and destination. However, actors and coordinators are transparent to each other. Hence, the dynamicity inherent in an actor system are hidden from the coordinators. Furthermore, as individual actors are grouped by roles based on their behaviors, coordination becomes much more scalable in systems of large scale. Figure 1 depicts the ARC model.

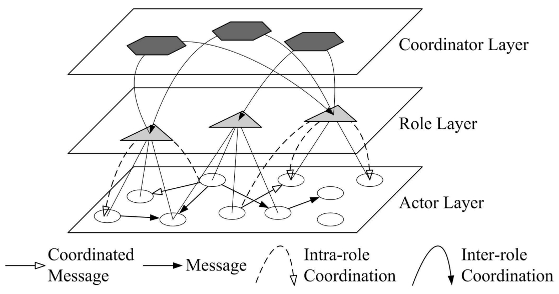


Fig. 1. The Actor-Role-Coordinator Model

From a coordinatee's perspective, coordination is exogenous and is distributed among roles and coordinators. In the same way as actors react to messages, roles and coordinators react to events. Both computation entities (actors) and coordination entities (roles and coordinators) emit events when their public states change. Based on observed events and the coordination invariants it is to maintain, a role not only makes decisions concerning its membership, but also makes decisions on message delivery time and location within the member set. The coordination is a composition of intra-role policies and inter-role constraints. The inter-role constraints are stored in distributed coordinators. If a role is constrained by multiple coordinators, the conjunction of the constraints from different coordinators must be satisfied. A similar situation exists for roles if an actor belongs to multiple roles. Partitioning the set of actors and minimizing the overlap of constraints between coordinators can reduce the complexity of an ARC system.

In the ARC model, the representation of constraints is built upon events which correspond to message dispatches. As an illustrative example, consider a sensor system consisting of three sensors and a decision unit that aggregates data sensed from the three sensors (e.g., by certain voting mechanisms). Clearly, the event that the decision unit aggregates the data must happen *after* the events that raw data from the three sensors are provided. Moreover, if we have consistency requirements on the data provided by the three sensors, we may constrain the *differences* between the occurrence times of the events that sensors provide their data (see Example 3.1 for detail). More specifically, we can treat precedence constraints and real-time constraints as coordination policies that enforce the following:

Precedence Constraints: Consider a distributed system with a set of observable events E. Precedence constraints of the form $e_i \prec e_j$ $(e_i, e_j \in E)$ restrict the occurrence of e_i to precede the occurrence of e_j.

Real-Time Constraints: Consider a real-time system with a set of observable events E. Timing constraints of the form $t(e_i) - t(e_j) \leq d$ $(e_i, e_j \in E$ and $d \in \Re^+ \cup \{+\infty\})$ restrict event e_i to occur no later than d time units after event e_j occurs.

Although temporal constraints and constraint satisfaction are studied extensively in the real-time community, such studies are from resource (such as processors) schedulability perspectives and have focused on specific types of constraints, rather than from programming language and constraint model perspectives.

Furthermore, as coordination constraints in the ARC model are distributed among coordinators and roles and these constraints are conjunctively applied on actors being constrained, it is essential that we have a uniform way to compose different sets of constraints and later be able to formally reason about the compositions and satisfiability.

3 The Semiring-Based Timing Constraint Models and Their Feasible Region Inclusions

As discussed in previous sections, coordination constraints can be distributed, but need to be conjunctively applied to the actors being constrained; thus for a pair of events, there can be multiple types of constraints imposed on them. For overlapping constraints, there often exists an implicit constraint derivable from the given constraint set that is a tighter constraint on the event pair than any of the explicitly specified ones. However, the existence of different constraint types complicates the derivation of implicit constraints. In this section, we unify precedence and real-time constraints in a semiring-based timing constraint model, utilize the all-pairs extremal paths algorithm on closed semirings to derive most stringent implicit constraints, and develop theories (inclusions and intersections) regarding the feasible regions of semiring-based timing constraints.

3.1 Semiring-Based Timing Constraints

Constraint semirings have been proposed as a framework for unifying QoS constraints [6]. A constraint semiring S is a tuple $(\langle A, \oplus, \otimes, \mathbf{0}, \mathbf{1}\rangle, \leq_S)$ where A is the carrier set and $\mathbf{0}, \mathbf{1} \in A$; $\oplus$ is commutative, associative, idempotent, and has $\mathbf{0}$ as its unit; $\otimes$ is commutative, associative, distributes over $\oplus$, and has $\mathbf{1}$ as its unit element and $\mathbf{0}$ as its absorbing element; and $\leq_S$ is a partial order induced by the idempotence of the $\oplus$ operation, i.e., $\forall a, b \in A : a \leq_S b$ iff $a \oplus b = b$. $\mathbf{0}$ is the minimum element of $\leq_S$ and $\mathbf{1}$ is the maximum element of $\leq_S$. The application of the framework in constraining transitions between states of a system or connectors between components can be found in [7] and [29], respectively. The following is an example of applying constraint semirings in coordinating actors.

Example 3.1 In the sensor system mentioned in Section 2, we assume that the corresponding events of the three sensor actors sending their data are e_1, e_2, and e_3, respectively. To guarantee the consistency of the votes, we constrain the differences between the occurrence times of the three events $t(e_1)$, $t(e_2)$, and $t(e_3)$ to be within certain ranges using real-time constraints as discussed above. The constraint set and its corresponding constraint matrix are given in (1)

$$\left\{\begin{array}{l} t(e_1)-t(e_2)\leq 6,\ \ t(e_2)-t(e_1)\leq 6,\\ t(e_1)-t(e_3)\leq 7,\ \ t(e_3)-t(e_1)\leq 3,\\ t(e_2)-t(e_3)\leq 9,\ \ t(e_3)-t(e_2)\leq 14 \end{array}\right\};\ \mathbf{D}^{(0)} = \begin{bmatrix} 0 & 6 & 7\\ 6 & 0 & 9\\ 3 & 14 & 0 \end{bmatrix} \tag{1}$$

where $\mathbf{D}^{(0)}$ is the constraint matrix indexed by the subscripts of events. For instance, because of the constraint $t(e_1) - t(e_2) \leq 6$, we have $d^{(0)}_{1,2} = 6$ in $\mathbf{D}^{(0)}$. The construction of $\mathbf{D}^{(0)}$ is given in (2). The constraint set can also be represented as a weighted directed constraint graph as shown in Fig. 2(a), from which implicit constraints can be derived by the Floyd-Warshall all-pairs shortest paths algorithm. Intuitively, in the given constraint set, the constraints $t(e_3) - t(e_1) \leq 3$ and $t(e_1) - t(e_2) \leq 6$ imply the constraint $t(e_3) - t(e_2) \leq 3 + 6 = 9$. Furthermore, this implied constraint is applied on the same pair of events as the constraint $t(e_3) - t(e_2) \leq 14$, hence we have $t(e_3) - t(e_2) \leq \min(14, 9) = 9$. Real-time constraints in this example can be naturally mapped into a constraint semiring $(\langle \Re^+ \cup \{+\infty\}, \min, +, +\infty, 0\rangle, \geq)$, where $\Re^+ \cup \{+\infty\}$ is the set of constraint values, $\min$ is used for parallel composition of two constraints in which both constrained events coincide; and $+$ is used for sequential composition of two constraints where there is a

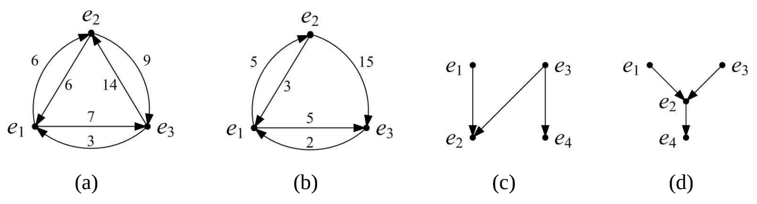


Fig. 2. Timing and precedence constraint graphs.

common event in both constraints differing in their signs. The $+\infty$ represents that (e_i, e_j) is not constrained [3], while 0 represents the most stringent constraint. The ordering relation $\geq$ on $\Re^+ \cup \{+\infty\}$ indicates the stringency of constraints, $a, b \in \Re^+ \cup \{+\infty\} : a \geq b \Leftrightarrow a \leq_S b$, i.e., the smaller the constraint value, the more stringent the constraint. In other words, we say the constraint $t(e_1) - t(e_2) \leq b$ is more stringent than the constraint $t(e_1) - t(e_2) \leq a$ if $a \geq b$ (or $a \leq_S b$). $\square$

Similarly, the constraint semiring $(\langle \{true, false\}, \vee, \wedge, false, true\rangle, \leq_S)$, in which $false \leq_S true$, can represent precedence constraints. Fig. 2(c) and 2(d) are examples of two different sets of precedence constraints, where the corresponding entry in the constraint matrix is $true$ if and only if $e_i \prec e_j$ (represented as $e_i \longrightarrow e_j$ in the figures) or $i = j$.

Given a set of initial constraints coming from different coordinators, it is important to know the implications of constraint compositions, i.e., the *implicit* constraints derivable from the given constraint sets. There are two scenarios that a new implicit constraint may arise, i.e., two given constraints are on the same pair of events (parallel edges in the constraint graph), or there is a common event in both constraints with different signs (connected edges in the constraint graph). These two scenarios corresponds to constraint parallel and sequential composition, respectively. The $\oplus$ and $\otimes$ operations of a semiring $\langle A, \oplus, \otimes, \mathbf{0}, \mathbf{1}\rangle$ are used for these operations accordingly. Under this model, the extremal paths algorithm on closed semirings [4] [15] can be directly applied to derive implicit constraints between all pairs of constrained events (Appendix A), where the initial matrix $\mathbf{D}^{(0)}$ on the set of external observables $e_i, i = 1, \ldots, n$ is given as [5]

$$d^{(0)}_{i,j} = \begin{cases} \text{the maximum element of } \leq_S & \text{if } i = j \\ \text{the constraint value of } (e_i, e_j) & \text{if } i \neq j \text{ and } (e_i, e_j) \text{ is constrained} \\ \text{the minimum element of } \leq_S & \text{if } i \neq j \text{ and } (e_i, e_j) \text{ is not constrained} \end{cases} \tag{2}$$

$d^{(n)}_{i,j}$ is the transitive closure of path length n between event e_i and e_j in the corresponding constraint graph and hence is the most stringent constraint (with respect to $\leq_S$ on the specific semiring) between events e_i and e_j derivable from the original set of constraints. We denote the all-pairs extremal paths matrix by $\mathbf{D}^*$.

For instance, in Example 3.1, the Floyd-Warshall algorithm for deriving implicit real-time constraint is a special case of Algorithm 1 (Appendix A) with $\oplus$ and $\otimes$ replaced by $\min$ and $+$, respectively; and $d^{(0)}_{i,i}, i = 1, \ldots, n$, are set to 0, the unit of $+$, and all the

[3] Constraints have directions. Therefore, the fact that (e_i, e_j) is not constrained does not imply that (e_j, e_i) is not constrained.

[4] A closed semiring requires that $\oplus$ and $\otimes$ are closed over A. As a counterexample, $\langle \Re \cup \{+\infty\}, \min, +, +\infty, 0\rangle$ is not closed since the summation of an infinite number of negative elements results in $-\infty$ which is not an element of $\Re \cup \{+\infty\}$. $\langle \Re \cup \{+\infty, -\infty\}, \min, +, +\infty, 0\rangle$ is also problematic as the unit element of $\min$, i.e., $+\infty$, is no longer the absorbing element of $+$, violating the definition of a semiring. In these cases, Algorithm 1 will not work.

[5] Without loss of generality, we assume that there is at most one constraint over each pair of events. If there are multiple constraints on an event pair (e_i, e_j), one can choose the most stringent constraint using the $\leq_S$ operation and drop the others.

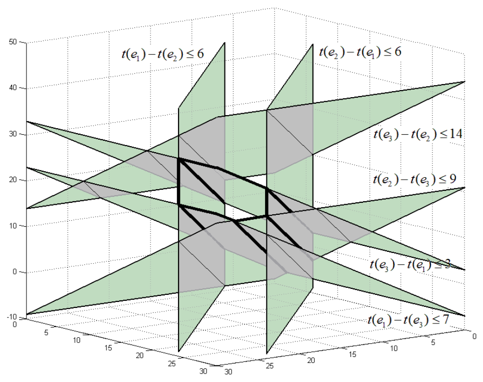
other unconstrained entries are set to $+\infty$, the unit of $\min$. Therefore, the most stringent constraints between all-pairs of events are given below:

$$\left\{\begin{array}{l} t(e_1)-t(e_2)\le 6,\ t(e_2)-t(e_1)\le 6, \\ t(e_1)-t(e_3)\le 7,\ t(e_3)-t(e_1)\le 3, \\ t(e_2)-t(e_3)\le 9,\ t(e_3)-t(e_2)\le 9 \end{array}\right\};\ \mathbf{D}^* = \mathbf{D}^{(3)} = \begin{bmatrix} 0 & 6 & 7 \\ 6 & 0 & 9 \\ 3 & 9 & 0 \end{bmatrix} \tag{3}$$

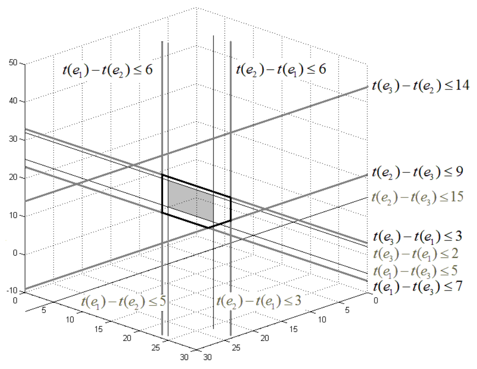
3.2 *Feasible Regions of Semiring-Based Timing Constraint Sets*

Coordination constraints eliminate otherwise possible computations of a system. Given two different sets of constraints C and C'. By showing that the computations allowed by C include those allowed by C', we avoid repeatedly checking computations against different coordination constraint sets. For instance, consider the two sets of precedence constraints as shown in Fig. 2(c) and 2(d), where $e_i \longrightarrow e_j$ indicates that $e_i \prec e_j$. Fig. 2(c) allows a trace set $T_c = \{e_1e_3e_2e_4, e_1e_3e_4e_2, e_3e_1e_2e_4, e_3e_1e_4e_2, e_3e_4e_1e_2\}$ [6]; and Fig. 2(d) allows a trace set $T_d = \{e_1e_3e_2e_4, e_3e_1e_2e_4\}$. Clearly, $T_d \subseteq T_c$. Therefore, if constraints in Fig. 2(c) result in message delivery orders that guarantee safety requirements, Fig. 2(d) will also guarantee the same properties.

Similarly, the timed trace of a real-time computation can be represented as a *timed data stream* [7] [5]. The set of all timed data streams satisfying a given set of real-time constraints is a convex set and we call the set the *feasible region* (of the set of real-time constraints) throughout the paper. For example, the feasible region of the set of real-time constraints given in (1) is illustrated in Fig. 3(a), with its boundaries marked as bold lines.



(a) The feasible region of constraint set (1).



(b) Inclusion of two feasible regions.

Fig. 3. Feasible region and feasible region inclusion. As can be seen from Fig. 3(a), each plane representing a constraint is parallel to the vector $\mathbf{z} = (-1)\mathbf{x_1} + (-1)\mathbf{x_2} + (-1)\mathbf{x_3}$, where vectors $\mathbf{x_1}$, $\mathbf{x_2}$, and $\mathbf{x_3}$ indicate time axes of events e_1, e_2, and e_3, respectively. Therefore, to facilitate the discussion of feasible region inclusion, we view the space in the direction of $\mathbf{z}$ in Fig. 3(b). We can see that the feasible region of constraint set (1) (gray bold lines) includes that of (4) (black light lines).

Now, consider another set of real-time constraints given in (4)

$$\left\{\begin{array}{l} t(e_1)-t(e_2)\le 5,\ t(e_2)-t(e_1)\le 3, \\ t(e_1)-t(e_3)\le 5,\ t(e_3)-t(e_1)\le 2, \\ t(e_2)-t(e_3)\le 15 \end{array}\right\};\ \mathbf{D}'^{(0)} = \begin{bmatrix} 0 & 5 & 5 \\ 3 & 0 & 15 \\ 2 & +\infty & 0 \end{bmatrix} \tag{4}$$

[6] Due to the synchronous event-based control mechanism of the ARC model mentioned in Section 2 and detailed in Section 4, event orders will indicate the corresponding message delivery orders. Also note that although we constrain only a predefined finite set of events, the complete trace with all events can be formed by permuting unconstrained events and the inclusion relation still holds. Moreover, given that the system is stabilized, such finite set of constrained events is obtainable.

[7] A timed data stream over an event set E is a pair (a, α) where a is a sequence with elements from E and α is a monotonically increasing sequence with elements from $\Re^+ \cup \{+\infty\}$.

The feasible region of the constraint set (4) can be shown to be included within that of (1) as illustrated in Fig. 3(b). Lemma 3.2, together with Theorem 3.3, shows that all-pairs shortest paths matrices of real-time constraint sets can be used for such comparison.

Lemma 3.2 *The feasible region of a set of real-time constraints does not change when constraints between all event pairs are replaced by implicit constraints derived from Algorithm 1 (Appendix A).*

Proof: The formal proof is given in Appendix B. □

For instance, the feasible region of (1) does not change when the constraint $t(e_3) - t(e_2) \leq 14$ is changed to $t(e_3) - t(e_2) \leq 9$.

Theorem 3.3 *Given two sets of real-time constraints C and C' on the same set of events*[8]. *Let their corresponding most stringent implicit constraint matrices (i.e., all-pairs shortest paths matrices) be $\mathbf{D}^*$ and $\mathbf{D}'^*$, respectively. The feasible region of C' is included within that of C if and only if $\mathbf{D}^* \geq \mathbf{D}'^* (\forall i,j : d^*_{i,j} \geq d'^*_{i,j})$ where $\geq$ is the ordering relation defined on the semiring $(\langle \Re^+ \cup \{+\infty\}, \min, +, +\infty, 0\rangle, \geq)$.*

Proof: The formal proof is given in Appendix C. □

This result can be easily extended to precedence constraints due to the following injection

$$(\langle \{true, false\}, \vee, \wedge, false, true\rangle, \leq_S) \overset{\substack{f(true)=0 \\ f(false)=+\infty}}{\longmapsto} (\langle \Re^+ \cup \{+\infty\}, \min, +, +\infty, 0\rangle, \geq) \qquad (5)$$

For example, the transitive closure matrices of the two sets of precedence constraints in Fig. 2(c) and 2(d) are

$$\mathbf{D}^* = \begin{bmatrix} true & true & false & false \\ false & true & false & false \\ false & true & true & true \\ false & false & false & true \end{bmatrix} \text{ and } \mathbf{D}'^* = \begin{bmatrix} true & true & false & true \\ false & true & false & true \\ false & true & true & true \\ false & false & false & true \end{bmatrix} \qquad (6)$$

respectively, where $d^*_{i,j}$ or $d'^*_{i,j}$ is $false$ if and only if $i \neq j$ and e_i does not precede e_j. Based on the ordering relation on $(\langle \{true, false\}, \vee, \wedge, false, true\rangle, \leq_S)$, we have $\mathbf{D}^* \leq_S \mathbf{D}'^*$ and thus the inclusion relation observed at the beginning of this section follows. For general semiring-based timing constraints, inclusion relations can be tested by

(i) applying Algorithm 1 with specific constraint semirings to get all-pairs extremal paths matrices of constraint sets; and

(ii) using the ordering relation $\leq_S$ on the constraint semiring to determine the dominant relationship between the all-pairs extremal paths matrices.

From Lemma 3.2, similar results can be given for *intersections* between feasible regions of timing constraint sets. An intersection of two constraint sets (not necessarily on the same set of events) can be used for deriving a constraint set that satisfies both sets of constraints. Such intersections are derived by forming the union of the constraint sets and applying Algorithm 1 with the corresponding constraint semirings. As the intersection of convex sets is still convex, similar proofs can be developed.

[8] Note that the event sets of the two constraint sets need not be the same in order for the two trace sets to be comparable. One can always extend both event sets to the same one by adding unconstrained events.

4 Integration of the ARC and Semiring-Based Timing Constraint Models through Maude

In this section, we use a canonical open system example, *the Restaurant for Dining Philosophers* [8], to illustrate the expressiveness of the ARC model and the integration of the ARC and the semiring-based timing constraint models.

Example 4.1 The Restaurant for Dining Philosophers: A restaurant has one table with n forks and n seats. Customers in the restaurant are m ($m > n$) philosophers who can be seated if there are free seats, and can stand up freeing the seat at any time if they have no fork. When seated, a philosopher eats if (s)he can grab two forks, otherwise (s)he thinks [9].

It is worth pointing out that the problem differs from the classical dining philosopher problem in that philosophers can freely join or leave the table at any time and hence introducing dynamicity and openness into the system. In addition, multiple constraints may co-exist. For example, constraints that avoid deadlock and constraints that give preferences to particular seats so that philosophers being seated there will always eat first.

The problem can be naturally expressed using the ARC model. More specifically, under the ARC model, philosophers and forks are actors. Two types of roles, i.e., seat roles and fork roles, are introduced to shield dynamicity from coordinators: n seat roles and n fork roles are circularly arranged as in the original problem of dining philosophers. Philosopher and fork actors can join and leave corresponding roles at any time. However, any role can only hold at most one actor at any instance of time. To simplify the presentation while still maintaining consistency of the model, we assume that fork actors are static, i.e., each fork role holds a fork actor and the membership does not change. On the other hand, the seat role's membership is dynamic in that its member philosopher changes frequently [10]. Multiple coordinators are introduced to impose coordination constraints on the roles so that properties such as deadlock free and preferences can be enforced.

In the remainder of this section, we detail the solution of the Restaurant for Dining Philosophers problem using the ARC model integrated with the semiring-based timing constraints. We use Maude [9], a tool that is well suited for specifying and verifying distributed systems, to write the specification and validate deadlock-free and preference properties.

4.1 Actors in Maude

In Maude, distributed system states are modeled as multisets (configurations) of actors and messages [9]. Configurations are formed by multiset union starting from singleton objects (actors) and messages. This is formalized by the following Maude declaration [11]

```
sort  Configuration .
subsorts Object Msg < Configuration .
op none : -> Configuration .
op __ : Configuration Configuration -> Configuration [ctor assoc comm id: none] .
```

[9] The requirement that $m > n$ is from the original problem in [8]. However, as will be seen, it is not to say that the table should be full before philosophers are allowed to eat, since the precedence constraints that avoid deadlock can in fact be fully distributed to each philosopher. However, inter-philosopher constraints, such as the preference constraints we introduce, may prevent certain philosophers in low priority seats from eating if the table is not full. But this does not cause livelock because philosophers are always free to move to the seat with higher priorities.

[10] Although each role has at most one member at any given time in this example, oftentimes, a role may have multiple actors.

[11] In Maude, `sorts` are used to declare types, the `subsort` relation on sorts parallels the subset relation on the sets of elements in the intended model of these sorts, an operator is declared with the keyword `op`, and `assoc`, `comm`, and `id` can be declared to specify equational axioms to denote associativity, commutativity, and identity, respectively. Also note that `Object` is used to represent actors throughout this paper.

A typical actor system configuration has the form

```
[actor_1] ... [actor_m] msg_1 ... msg_n
```

Each actor has an `id`, a set of `attributes`, and `in` and `out` queues for buffering incoming and outgoing messages. In other words, an actor object has the form

```
[id : cid | attributes | in: inQ, out: outQ ]
```

In actor systems, the message order is not specified and a message can be delivered *at any time* as long as its target matches a receiving actor as shown in the following rewrite rule (`rl`) for message delivery

```
rl[in] :
  [id : cid | attributes | in: inQ, out: outQ ] msg(id, id', cv)  =>
  [id : cid | attributes | in: (inQ, msg(id, id', cv)), out: outQ ] .
```

Similarly, the following rewriting rule states that a message is sent when it is at the head of an actor's output queue.

```
rl[out] :
  [id : cid | attributes | in: inQ, out: (msg(id', id, cv), outQ) ]  =>
  [id : cid | attributes | in: inQ, out: outQ ] msg(id', id, cv) .
```

Without coordination constraints, the initial configuration of the restaurant for dining philosophers system in Maude is the following

```
[o("p-i"): Phil | status: 1, R:(o("f-i"), 0), L:(o("f-j"), 0) | in: nil, out: nil]
[o("f-i"): Fork | acquired?: false | in: nil, out: nil]
```

where $\mathtt{i} = 1, \ldots, n$, and $\mathtt{j} = \mathtt{i} + 1$ if $\mathtt{i} \neq n$ and 1 if $\mathtt{i} = n$. A philosopher's `status` indicates if (s)he is waiting to be seated (`0`), seated and thinking (`1`), waiting for both forks (`2`), or eating (`3`); and attributes in `R`/`L` indicate the philosopher's right/left fork actor's `id` and current status of the fork (`0` for "no request sent", `1` for "request message sent", `2` for "fork acquired", and `3` for "release message sent"), respectively.

It is clear that in the above specification, the openness and dynamicity are not supported as philosopher actors need to explicitly know the names of their left and right fork actors. Therefore, if philosophers are allowed to leave, join, or move, they will not know the correct fork actors to send the request or release messages. Moreover, a deadlock configuration such as the following (when $m = n = 3$)

```
[o("p1"): Phil | status: 2, R:(o("f1"), 2), L:(o("f2"), 1) | in: nil, out: nil]
[o("p2"): Phil | status: 2, R:(o("f2"), 2), L:(o("f3"), 1) | in: nil, out: nil]
[o("p3"): Phil | status: 2, R:(o("f3"), 2), L:(o("f1"), 1) | in: nil, out: nil]
[o("f1"): Fork | acquired?: true | in: (msg(o("f1"),o("p2"),"request")), out: nil]
[o("f2"): Fork | acquired?: true | in: (msg(o("f2"),o("p3"),"request")), out: nil]
[o("f3"): Fork | acquired?: true | in: (msg(o("f3"),o("p1"),"request")), out: nil]
```

can be reached where `p1` holds `f1` requesting for `f2`, `p2` holds `f2` requesting for `f3`, and `p3` holds `f3` requesting for `f1`. In Maude, deadlock configurations can be found by the `search` command. Hence, a level of abstraction is needed to allow dynamicity and coordination constraints are necessary in order to avoid deadlock.

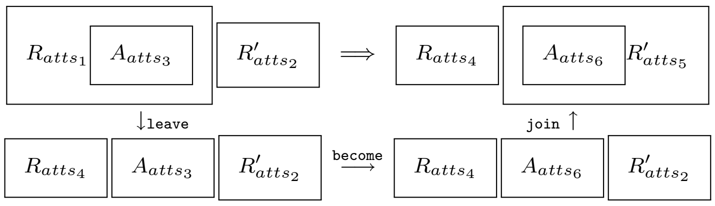
4.2 Roles in Maude

Roles are modeled in Maude as a special case of the Reflective Russian Doll (RRD) [18] model in which distributed states are nested and can be seen as a distributed soup of soups instead of a flat soup of actors and messages. The two level nested configuration, in the ARC case, consists of roles (meta-level objects) and role messages (meta-level messages) with roles' configurations consisting of coordinated actors (base-level objects) and actor messages (base-level messages). A role has the form

```
[ rid : cid | attributes, {configuration} | in: inQ, out: outQ ]
```

where `configuration` is a flat soup of actors and messages. There are three primitives defined in a role, i.e., `membership-change`, `up`, and `down`; and their corresponding conditional rewrite rules (`crl`) are presented informally as following

- `crl[membership-change]` guarantees that each actor may play one and only one role at any time. In order for an actor to change its role membership, it `leave`s a role R (causing changes in the state of R), `become`s an actor with another behavior (causing changes in the state of itself), and `join`s another role R'(causing changes in the state of R'). The `leave`, `become`, and `join` operations must be done atomically to avoid dangling actors.



- `crl[up]` addresses the *openness* issue: it extracts a message from the configuration in a role to the role's output queue. Since actors are sometimes anonymous to each other in open systems, a role is responsible for rerouting a message sent by an actor under it to a proper destination role.
- `crl[down]` addresses the *intra-role coordination* issue: it dequeues a message from a role's input queue and puts it into the role's configuration. Since actors under a role share common behaviors and also have diversities, a role is responsible for choosing a proper actor or proper actors for processing the message sent to it.

The specific policies for rerouting messages used in `crl[up]` and `crl[down]` should be defined in the specific role instances. When roles are added, coordination is based on roles rather than based on specific actors.

In the Restaurant for Dining Philosophers problem, roles can be used to model "seats of philosophers" to address openness and dynamicity since "seats" are stable. Now, the initial configuration for the system becomes

```
[o("default"): DefaultRole | {
  [o("p-k"): Phil | status: 0, R:(o("n/a"), 0), L:(o("n/a"), 0) | ...]
} | ...]
[o("S-i"): SeatRole | occupied: false, R: o("F-i"), L: o("F-j"), { none } | ...]
[o("F-i"): ForkRole | { [o("f-i"): Fork | acquired?: false | ...] } | ...]
```

where $\mathtt{i} = 1, \ldots, n$, $\mathtt{j} = \mathtt{i} + 1$ if $\mathtt{i} \neq n$ and 1 if $\mathtt{i} = n$, $\mathtt{k} = 1, \ldots, m$ and every occurrence of "`in: nil, out: nil`" is replaced with "`...`" for simplicity. The `DefaultRole` contains actors waiting to be seated (`status:0`). The *atomic* role membership change rule `crl[membership-change]` as well as `become` of a `Phil`, `join` of a `SeatRole`, and `leave` of a `DefaultRole` ensure that when a philosopher changes its `status` to `1` (thinking), it can be seated in some `SeatRole` as long as the role's `occupied` attribute is `false` (which changes to `true` atomically); and the mechanism for a philosopher to leave a seat is similar. Also note that a philosopher now does not need to know its left and right forks; `SeatRole` will reroute a message to the correct `ForkRole` based on its `R` and `L` attributes. For example, `msg(o("n/a"),o("p1"),"request")` in the actor level soup will be rerouted as `msg(o("F1"),o("S1"),"request")` in the role level soup. The `SeatRole` records the necessary information to handle the reply messages. As can be seen, the openness and dynamicity issues are solved using roles without any changes of the original actors defined.

4.3 Coordinators with Semiring-Based Timing Constraints in Maude

In the Restaurant for Dining Philosophers problem, we also need constraints that avoid deadlock and achieve preference requirements. A classic solution that avoids deadlock is to break the symmetry by having each philosopher first grab a fork with the lower number. This can be done by restricting `msg(o("S1"),o("F1"),"available")` (`o("F1")`'s reply to `msg(o("F1"),o("S1"),"request")` if `o("F1")` has not been acquired) to be delivered before `msg(o("F2"),o("S1"),"request")`.

Furthermore, a preference constraint that favors the philosopher sitting in `o("S1")` can be enforced by restricting `"available"` messages to `o("S1")` delivered before `"request"` messages from all the other `SeatRole`s. In the following, we discuss how constraints are enforced through exogenous event-based message controls by coordinators.

4.3.1 Semiring-Based Timing Constraints in Maude

In Maude, the concept of semiring is defined as a functional theory [9] and Algorithm 1 (Appendix A) can be defined as a parameterized functional module over a general semiring where `fmod MATRIX` implements Algorithm 1 (`op APXP`).

```
fmod MATRIX{X :: SEMIRING} is
  pr (ARRAY * (sort Entry{X,Y} to Entry{Y}, sort Array{X,Y} to Matrix{Y}))
  {IndexPair, X} .
  op APXP(_,_) : Matrix{X} Nat -> Matrix{X} .
  ...... ***omitted due to page limit
endfm
```

The partial order constraint model can thus be defined as the `view` from a general semiring $\langle A, \oplus, \otimes, \mathbf{0}, \mathbf{1}\rangle$ to Boolean algebra $\langle \{false, true\}, \vee, \wedge, false, true\rangle$ by mapping A, $\oplus$, $\otimes$, $\mathbf{0}$, and $\mathbf{1}$ to $\{false, true\}$, $\vee$, $\wedge$, $false$, and $true$, respectively.

```
view BOOL-SEMIRING from SEMIRING to BOOL is
  sort Elt to Bool .
  op 1 to term true .
  op 0 to term false .
  op X:Elt * Y:Elt to term X:Bool and Y:Bool .
  op X:Elt + Y:Elt to term X:Bool or Y:Bool .
endv
```

and the corresponding constraint matrices can be defined by making `fmod MATRIX` take the parameter of the specific semiring `BOOL-SEMIRING`

```
fmod BOOL-SEMIRING-MATRIX is
  protecting MATRIX{BOOL-SEMIRING} *
            (sort Entry{BOOL-SEMIRING} to BoolMatrixEntry,
             sort Matrix{BOOL-SEMIRING} to BoolMatrix,
             op empty to zeroMatrix) .
endfm
```

Real-time constraints over $\langle \Re^{+} \cup \{+\infty\}, \min, +, +\infty, 0\rangle$ can be defined similarly.

4.3.2 Coordinators in Maude

Without coordination, actors/roles follow the communication mechanisms `rl[in]` and `rl[out]` as in Section 4.1. However, with exogenous coordination, the corresponding event `in(id, id', cv)` or `out(id, id', cv)` of a message `msg(id, id', cv)` must be synchronously tested against the (presumably unique) coordinator for consistency before it can be delivered to the target actor. For instance, given the timing constraint model, under *partial order* constraints, a coordinator is a quadruple `[APXP(M) | eset | emap | n]`, where `M` is the initial constraint matrix indexed by `1` through `n`, `eset` is the set of indices of events of interest that have occurred (and satisfy the constraints in `M`), `emap` is the mapping from the set of events of interest to the set of indices, and `n` is the number of events of interest (also the dimension of `M`). Therefore, messages cannot be delivered freely as in `rl[in]` when a coordinator is present; instead, in order for a message to be delivered to its target actor, we first need to check if the corresponding event of message delivery is constrained and deliver the message if it is not.

```
crl[in-uncoord] :
  [id : cid | atts | in: inQ, out: outQ] msg(id, id’, cv) [M | eset | emap | n]
  =>
  [id : cid | atts | in: (inQ, msg(id, id’, cv)), out: outQ] [M | eset | emap | n]
  if emap[in(id, id’, cv)] == undefined .
```

When the event is constrained, we must check if the event satisfies all constraints in `M`

```
crl[in-coord] :
  [id : cid | atts | in: inQ, out: outQ] msg(id, id’, cv) [M | eset | emap | n]
  =>
  [id : cid | atts | in: (inQ, msg(id, id’, cv)), out: outQ]
  [M | insert(emap[in(id, id’, cv)], eset) | emap | n]
  if (tell([M | eset | emap | n], in(id, id’, cv))) .
```

where `op tell([M | eset | emap | n], e)` decides if all the predecessors of `emap[e]` have already occurred (in `eset`)[12]. The coordination mechanism for `out` is defined symmetrically as `in`.

In the Restaurant for Dining Philosophers problem, to avoid deadlock, we only need to put the following coordinator in the soup of roles defined above. This results in the following initial configuration (when $m = 4$ and $n = 3$)

```
[ APXP([1,2] |-> true ; [3,4] |-> true ; [5,6] |-> true, 6) | empty
  | (in(o("S1"), o("F1"), "available")|-> 1, out(o("F2"), o("S1"), "request")|-> 2,
     in(o("S2"), o("F2"), "available")|-> 3, out(o("F3"), o("S2"), "request")|-> 4,
     in(o("S3"), o("F1"), "available")|-> 5, out(o("F3"), o("S3"), "request")|-> 6)
  | 6 ]

[o("default"): DefaultRole | {
  [o("p1"): Phil | status: 0, R:(o("n/a"), 0), L:(o("n/a"), 0) | ...]
  [o("p2"): Phil | status: 0, R:(o("n/a"), 0), L:(o("n/a"), 0) | ...]
  [o("p3"): Phil | status: 0, R:(o("n/a"), 0), L:(o("n/a"), 0) | ...]
  [o("p4"): Phil | status: 0, R:(o("n/a"), 0), L:(o("n/a"), 0) | ...]
} | ...]

[o("S1"): SeatRole | occupied: false, R: o("F1"), L: o("F2"), { none } | ...]
[o("S2"): SeatRole | occupied: false, R: o("F2"), L: o("F3"), { none } | ...]
[o("S3"): SeatRole | occupied: false, R: o("F3"), L: o("F1"), { none } | ...]
[o("F1"): ForkRole | { [o("f1"): Fork | acquired?: false | ...] } | ...]
[o("F2"): ForkRole | { [o("f2"): Fork | acquired?: false | ...] } | ...]
[o("F3"): ForkRole | { [o("f3"): Fork | acquired?: false | ...] } | ...]
```

and the `search` for deadlock configurations in Maude finds no solution, indicating that the constraints have avoided the deadlock. Moreover, preference constraints can be enforced through the following coordinator

```
[ APXP([1,2] |-> true ; [3,4] |-> true , 4) | empty
  | (in(o("S1"), o("F1"), "available")|-> 1, out(o("F1"), o("S3"), "request")|-> 2,
     in(o("S1"), o("F2"), "available")|-> 3, out(o("F2"), o("S2"), "request")|-> 4)
  | 4 ]
```

and the `search` for configurations where philosophers in seat `S2` or `S3` eat before the philosopher in seat `S1` finds no solution. Moreover, this coordinator can be combined with the deadlock-avoidance coordinator by intersecting their constraints as discussed in Section 3. More specifically, the following coordinator

```
[ APXP([1,2] |-> true ; [3,4] |-> true ; [5,6] |-> true ;
       [1,9] |-> true ; [7,8] |-> true , 9) | empty
  | (in(o("S1"), o("F1"), "available")|-> 1, out(o("F2"), o("S1"), "request")|-> 2,
     in(o("S2"), o("F2"), "available")|-> 3, out(o("F3"), o("S2"), "request")|-> 4,
     in(o("S3"), o("F1"), "available")|-> 5, out(o("F3"), o("S3"), "request")|-> 6,
     in(o("S1"), o("F2"), "available")|-> 7, out(o("F2"), o("S2"), "request")|-> 8,
     out(o("F1"), o("S3"), "request") |-> 9)
  | 9 ]
```

[12]One can easily extend coordinators to constrain recurring events or event types, as opposed to single events, by adding sequence numbers to events of the same type in the event history `eset`. However, this makes the `search` space infinite and we restrict our discussion to single events.

is composed of the two coordinators by forming the intersection of the the two constraint sets. The trace set of the intersection can be easily shown to be included within trace sets of both constraint sets based on Theorem 3.3. Therefore, we can guarantee that both the deadlock-free and the preference requirements are met without having to repeatedly `search` all possible configurations.

5 Conclusion

This paper presents a continuation of our previous work on the Actor-Role-Coordinator model for asynchronous open distributed and embedded systems. We focus on the way that actor or role messages are manipulated exogenously based on precedence and real-time constraints imposed on their corresponding events. We discuss the important properties of semiring-based timing constraints that generalize different timing constraint types. More specifically, we apply the all-pairs extremal paths algorithm on closed semirings to derive comparable forms of timing constraint sets which allow us to decide inclusions and find intersections between feasible regions of timing constraint sets. To illustrate the way exogenous coordinations, i.e., behavior abstractions by roles and computation restrictions by coordinators, are imposed on actor systems, we present the ARC solution to a canonical open system problem, the Restaurant for Dining Philosophers problem. We specify and integrate these coordinating entities through Maude rewriting logic language, and are able to show that the coordination requirements are met through Maude's verification tools and the properties we give for semiring-based timing constraints.

Note that we have not yet applied our theories to systems with real-time constraints; neither have we presented compositions of different types of constraints. Our future work thus targets the utilization of the semiring-based timing constraint model in systems with real-time constraints or different types of constraints. For example, in the Restaurant for Dining Philosophers problem, a typical real-time constraint could stipulate that philosophers release their forks before d time units after they acquire them. However, incorporating real-time constraints would require us to prototype the system through Real-Time Maude [19]. In Real-Time Maude, time is originally modeled as an ordered commutative monoid $\langle \texttt{Time}, +, 0\rangle$. The inclusion of the `min` operator and its unit `INF` in `NAT-TIME-DOMAIN-WITH-INF` has made time a semiring, which coincides the constraint model we implemented in Section 4. Moreover, the inclusion and intersection of semiring-based timing constraints discussed in Section 3 provide us a basis to study *similarities* between timing constraints which are important in comparing imprecise systems.

References

[1] Agha, G., I. A. Mason, S. F. Smith and C. L. Talcott, *A foundation for actor computation*, Journal of Functional Programming **7** (1997), pp. 1–72.

[2] Arbab, F., *IWIM: A communication model for cooperative systems*, in: *Proceedings of the 2nd International Conference on the Design of Cooperative Systems*, 1996, pp. 567–585.

[3] Arbab, F., *A channel-based coordination model for component composition*, Technical report, Centrum voor Wiskunde en Informatica, Amsterdam (2001).

[4] Arbab, F., *Abstract behavior types: a foundation model for components and their composition*, Science of Computer Programming **55** (2005), pp. 3–52.

[5] Arbab, F. and J. Rutten, *A coinductive calculus of component connectors*, in: *Proceedings of the 16th International Workshop on Algebraic Development Techniques*, LNCS **2755**, 2002, pp. 34–55.

[6] Bistarelli, S., U. Montanari and F. Rossi, *Semiring-based constraint satisfaction and optimization*, Journal of the ACM **44** (1997), pp. 201–236.

[7] Chothia, T. and J. Kleijn, *Q-automata: Modelling the resource usage of concurrent components*, in: *Proceedings of the 5th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA 2006)*, ENTCS **175**, 2007, pp. 153–167.

[8] Ciancarini, P., *Coordination languages for open system design*, in: *Proceedings of the International Conference on Computer Languages*, 1990, pp. 252–260.

[9] Clavel, M., F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer and C. Talcott, "All About Maude: A High-Performance Logical Framework," Springer, 2007.

[10] Colman, A. and J. Han, *Coordination systems in role-based adaptive software*, in: *Proceedings of the 7th International Conference on Coordination Models and Languages*, LNCS **3454**, 2005, pp. 63–78.

[11] Cruz, J.-C. and S. Ducasse, *CoLaS: A group based approach for coordinating active objects*, in: *Proceedings of the 3rd International Conference on Coordination Languages and Models*, LNCS **1594**, 1999, pp. 355–371.

[12] De Nicola, R., G. L. Ferrari, U. Montanari, R. Pugliese and E. Tuosto, *A process calculus for QoS-aware applications*, in: *Proceedings of the 7th International Conference on Coordination Models and Languages*, 2005, pp. 33–48.

[13] De Nicola, R., G. L. Ferrari and R. Pugliese, *KLAIM: a kernel language for agents interaction and mobility*, IEEE Transactions on Software Engineering (Special Issue on Mobility and Network Aware Computing) **24** (1998), pp. 315–330.

[14] De Nicola, R., J.-P. Katoen, D. Latella and M. Massink, *Towards a logic for performance and mobility*, in: *Proceedings of the 3rd Workshop on Quantitative Aspects of Programming Languages, QAPL05*, ENTCS **153**, 2005, pp. 161–175.

[15] Fletcher, J. G., *A more general algorithm for computing closed semiring costs between vertices of a directed graph*, Communications of the ACM **23** (1980), pp. 350–351.

[16] Gelernter, D., *Generative communication in Linda*, TOPLAS **7** (1985), pp. 80–112.

[17] Jacquet, J.-M., K. D. Bosschere and A. Brogi, *On timed coordination languages*, in: *Proceedings of the 4th International Conference on Coordination Languages and Models*, LNCS **1906**, 2000, pp. 81–98.

[18] Meseguer, J. and C. L. Talcott, *Semantic models for distributed object reflection*, in: *Proceedings of the 16th European Conference on Object-Oriented Programming*, LNCS **2374**, 2002, pp. 1–36.

[19] Ölveczky, P. C., "Real-Time Maude 2.3 Manual," (2007).

[20] Omicini, A. and E. Denti, *Formal* ReSpecT, in: A. Dovier, M. C. Meo and A. Omicini, editors, *Declarative Programming – Selected Papers from AGP'00*, ENTCS **48**, 2001 pp. 179–196.

[21] Omicini, A. and E. Denti, *From tuple spaces to tuple centres*, Science of Computer Programming **41** (2001), pp. 277–294.

[22] Omicini, A., A. Ricci and M. Viroli, *Formal specification and enactment of security policies through agent coordination contexts*, in: *Proceedings of the 1st International Workshop on Security Issues in Coordination Models, Languages, and Systems*, ENTCS **85**, 2003, pp. 17–36.

[23] Papadopoulos, G. A. and F. Arbab, *Coordination of systems with real-time properties in manifold*, in: *Proceedings of the 20th Conference on Computer Software and Applications* (1996), p. 50.

[24] Papadopoulos, G. A. and F. Arbab, *Coordination models and languages*, Technical report, Centrum voor Wiskunde en Informatica, Amsterdam (1998).

[25] Picco, G., A. Murphy and G.-C. Roman, *LIME: Linda meets mobility*, in: *Proceedings of the 21st International Conference on Software Engineering*, 1999, pp. 368–377.

[26] Ren, S., Y. Yu, N. Chen, K. Marth, P.-E. Poirot and L. Shen, *Actors, roles and coordinators - a coordination model for open distributed and embedded systems.*, in: *Proceedings of the 8th International Conference on Coordination Models and Languages*, LNCS **4038**, 2006, pp. 247–265.

[27] Saraswat, V., R. Jagadeesan and V. Gupta, *Foundations of timed concurrent constraint programming*, in: *Proceedings of the 9th Annual IEEE Symposium on Logic in Computer Science*, 1994, pp. 71–80.

[28] Talcott, C., M. Sirjani and S. Ren, *Comparing three coordination models: Reo, ARC, and RRD*, in: *Proceedings of the 6th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA 2007)*, ENTCS **194**, 2007, pp. 39–55.

[29] Zhou, Y. and E. A. Lee, *A causality interface for deadlock analysis in dataflow*, in: *Proceedings of the 6th ACM & IEEE International conference on Embedded software* (2006), pp. 44–52.

Appendix

A The All-Pairs Extremal Paths Algorithm on Closed Semirings

Algorithm 1 ALL-PAIRS-EXTREMAL-PATHS

```
1: for $k$ = 1 to $n$ do
2:     for $i$ = 1 to $n$ do
3:         for $j$ = 1 to $n$ do
4:             $d_{i,j}^{(k)} = d_{i,j}^{(k-1)} \oplus (d_{i,k}^{(k-1)} \otimes d_{k,j}^{(k-1)})$
5:         end for
6:     end for
7: end for
```

B Proof of Lemma 3.2

Lemma 3.2 Given a set of m timing constraints of the form $t(e_i) - t(e_j) \leq d_k$ among n events, $\mathbf{At} \leq \mathbf{d}$, where $\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{t} = \begin{bmatrix} t(e_1) & \dots & t(e_n) \end{bmatrix}^{\mathrm{T}}$, and $\mathbf{d} = \begin{bmatrix} d_1 & \dots & d_m \end{bmatrix}^{\mathrm{T}}$. We have $\{\mathbf{t}\,|\mathbf{At} \leq \mathbf{d}\} = \left\{\mathbf{t}\,\middle|\widetilde{\mathbf{A}}\mathbf{t} \leq \widetilde{\mathbf{d}}\right\}$, i.e., the set of solutions of $\mathbf{At} \leq \mathbf{d}$ is the same as the set of solutions of $\widetilde{\mathbf{A}}\mathbf{t} \leq \widetilde{\mathbf{d}}$ where

$$\widetilde{\mathbf{A}} = \left[\begin{array}{ccccc} 1 & -1 & & & \\ 1 & & -1 & & \\ \vdots & & & \ddots & \\ 1 & & & & -1 \\ \hline -1 & 1 & & & \\ & 1 & -1 & & \\ & \vdots & & \ddots & \\ & 1 & & & -1 \\ \hline \vdots & & \dots & & \vdots \\ \hline -1 & & & & 1 \\ & -1 & & & 1 \\ & & \ddots & & \vdots \\ & & & -1 & 1 \end{array}\right] \quad \text{and} \quad \widetilde{\mathbf{d}} = \left[\begin{array}{c} d^*_{1,2} \\ d^*_{1,3} \\ \vdots \\ d^*_{1,n} \\ \hline d^*_{2,1} \\ d^*_{2,3} \\ \vdots \\ d^*_{2,n} \\ \hline \vdots \\ \hline d^*_{n,1} \\ d^*_{n,2} \\ \vdots \\ d^*_{n,n-1} \end{array}\right] \tag{B.1}$$

and $d^*_{i,j}, i \neq j$ are the shortest path weights.

Proof:

(i) $\{\mathbf{t}\,|\mathbf{At} \leq \mathbf{d}\} \supseteq \left\{\mathbf{t}\,\middle|\widetilde{\mathbf{A}}\mathbf{t} \leq \widetilde{\mathbf{d}}\right\}$

This directly follows from the fact that $\mathbf{A}$ *contains some rows of* $\widetilde{\mathbf{A}}$ *and the corresponding* d*'s in* $\mathbf{d}$ *is no less than those in* $\widetilde{\mathbf{d}}$ *(the shortest path weights).*

(ii) $\{\mathbf{t}\,|\mathbf{At} \leq \mathbf{d}\} \subseteq \left\{\mathbf{t}\,\middle|\widetilde{\mathbf{A}}\mathbf{t} \leq \widetilde{\mathbf{d}}\right\}$

Assume to the contrary that there is a vector $\mathbf{t}' = \begin{bmatrix} t_1 & \dots & t_n \end{bmatrix}^{\mathrm{T}}$ *s.t.* $\mathbf{t}' \in \{\mathbf{t}\,|\mathbf{At} \leq \mathbf{d}\} \wedge$

$\mathbf{t}' \notin \left\{\mathbf{t} \,\middle|\, \widetilde{\mathbf{A}}\mathbf{t} \leq \widetilde{\mathbf{d}}\right\}$. *This implies that the following set of linear inequalities has no solution*

$$\begin{bmatrix} \mathbf{I} \\ -\mathbf{I} \\ \widetilde{\mathbf{A}} \end{bmatrix} \mathbf{t} \leq \begin{bmatrix} \mathbf{t}' \\ -\mathbf{t}' \\ \widetilde{\mathbf{d}} \end{bmatrix} \tag{B.2}$$

Based on Farkas' Lemma, together with the infeasibility of (B.2), we have that there exists an (n^2+n)*-vector* $\left[\mathbf{t}_1^{\mathrm{T}}\ \mathbf{t}_2^{\mathrm{T}}\ \mathbf{t}_3^{\mathrm{T}}\right]^{\mathrm{T}}$ *where* $\mathbf{t}_1$ *and* $\mathbf{t}_2$ *are two* n*-vector and* $\mathbf{t}_3^{\mathrm{T}}$ *is a* (n^2-n)*-vector, such that (B.3), (B.4), and (B.5) hold*

$$\begin{bmatrix} \mathbf{I} & -\mathbf{I} & \widetilde{\mathbf{A}}^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \mathbf{t}_1 \\ \mathbf{t}_2 \\ \mathbf{t}_3 \end{bmatrix} = \mathbf{0} \tag{B.3}$$

$$\begin{bmatrix} \mathbf{t}_1 \\ \mathbf{t}_2 \\ \mathbf{t}_3 \end{bmatrix} \geq \mathbf{0} \tag{B.4}$$

$$\begin{bmatrix} \mathbf{t}'^{\mathrm{T}} & -\mathbf{t}'^{\mathrm{T}} & \widetilde{\mathbf{d}}^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \mathbf{t}_1 \\ \mathbf{t}_2 \\ \mathbf{t}_3 \end{bmatrix} < 0 \tag{B.5}$$

From (B.3) we have that

$$\mathbf{t}_1 - \mathbf{t}_2 = -\widetilde{\mathbf{A}}^{\mathrm{T}}\mathbf{t}_3 \tag{B.6}$$

Insert (B.6) into (B.5) we have that

$$-\mathbf{t}'^{\mathrm{T}}\widetilde{\mathbf{A}}^{\mathrm{T}}\mathbf{t}_3 + \widetilde{\mathbf{d}}^{\mathrm{T}}\mathbf{t}_3 = \left(\widetilde{\mathbf{d}}^{\mathrm{T}} - \mathbf{t}'^{\mathrm{T}}\widetilde{\mathbf{A}}^{\mathrm{T}}\right)\mathbf{t}_3 < 0 \tag{B.7}$$

Therefore, it must be that

$$\exists i,j : d^*_{i,j} < t_i - t_j \tag{B.8}$$

since otherwise $\widetilde{\mathbf{d}}^{\mathrm{T}} - \mathbf{t}'^{\mathrm{T}}\widetilde{\mathbf{A}}^{\mathrm{T}} \geq \mathbf{0}$ *together with (B.4) would imply* $\left(\widetilde{\mathbf{d}}^{\mathrm{T}} - \mathbf{t}'^{\mathrm{T}}\widetilde{\mathbf{A}}^{\mathrm{T}}\right)\mathbf{t}_3 \geq 0$ *which contradicts (B.7). However, (B.8) contradicts the fact that* $d^*_{i,j}$ *is the optimal solution to the linear program*

$$\begin{aligned} \textbf{maximize}\quad & t(e_i) - t(e_j) \\ \textbf{subject to}\quad & \mathbf{A}\mathbf{t} \leq \mathbf{d} \end{aligned} \tag{B.9}$$

i.e., $d^*_{i,j}$ *is the shortest path weight. Therefore, we have* $\{\mathbf{t}\,|\mathbf{A}\mathbf{t} \leq \mathbf{d}\} \subseteq \left\{\mathbf{t} \,\middle|\, \widetilde{\mathbf{A}}\mathbf{t} \leq \widetilde{\mathbf{d}}\right\}$ *and thus* $\{\mathbf{t}\,|\mathbf{A}\mathbf{t} \leq \mathbf{d}\} = \left\{\mathbf{t} \,\middle|\, \widetilde{\mathbf{A}}\mathbf{t} \leq \widetilde{\mathbf{d}}\right\}$. □

C Proof of Theorem 3.3

Theorem 3.3 Given two timing constraint sets $\mathbf{At} \leq \mathbf{d}$, and $\mathbf{A}'\mathbf{t} \leq \mathbf{d}'$, the convex polyhedron of $\mathbf{At} \leq \mathbf{d}$ is included in the convex polyhedron of $\mathbf{A}'\mathbf{t} \leq \mathbf{d}'$ if and only if $\widetilde{\mathbf{d}} \leq \widetilde{\mathbf{d}}'$ for $\widetilde{\mathbf{A}}\mathbf{t} \leq \widetilde{\mathbf{d}}$, and $\widetilde{\mathbf{A}}\mathbf{t} \leq \widetilde{\mathbf{d}}'$, where $\widetilde{\mathbf{A}}$, $\widetilde{\mathbf{d}}$, and $\widetilde{\mathbf{d}}'$ are defined as in Lemma 3.2.

Proof:

Note that the convex polyhedron of $\mathbf{At} \leq \mathbf{d}$ *is included in the convex polyhedron of* $\mathbf{A}'\mathbf{t} \leq \mathbf{d}'$ *if and only if the convex polyhedron of* $\begin{bmatrix}\mathbf{A}^{\mathrm{T}} & \mathbf{A}'^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}} \mathbf{t} \leq \begin{bmatrix}\mathbf{d}^{\mathrm{T}} & \mathbf{d}'^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$ *is the convex polyhedron of* $\mathbf{At} \leq \mathbf{d}$. *Hence, we prove that* $\widetilde{\mathbf{d}} \leq \widetilde{\mathbf{d}}'$ *if and only if* $\begin{bmatrix}\mathbf{A}^{\mathrm{T}} & \mathbf{A}'^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}} \mathbf{t} \leq \begin{bmatrix}\mathbf{d}^{\mathrm{T}} & \mathbf{d}'^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$ *is the convex polyhedron of* $\mathbf{At} \leq \mathbf{d}$.

(i) Necessary condition:

Suppose we have $\widetilde{\mathbf{d}} \leq \widetilde{\mathbf{d}}'$, *it is easy to see that* $\begin{bmatrix}\widetilde{\mathbf{A}}^{\mathrm{T}} & \widetilde{\mathbf{A}}^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}} \mathbf{t} \leq \begin{bmatrix}\widetilde{\mathbf{d}}^{\mathrm{T}} & \widetilde{\mathbf{d}}'^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$ *has the same solution set as* $\widetilde{\mathbf{A}}\mathbf{t} \leq \widetilde{\mathbf{d}}$. *Therefore, from Lemma 3.2,* $\begin{bmatrix}\mathbf{A}^{\mathrm{T}} & \mathbf{A}'^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}} \mathbf{t} \leq \begin{bmatrix}\mathbf{d}^{\mathrm{T}} & \mathbf{d}'^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$ *has the same solution set as* $\mathbf{At} \leq \mathbf{d}$.

(ii) Sufficient condition:

Assume $\begin{bmatrix}\mathbf{A}^{\mathrm{T}} & \mathbf{A}'^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}} \mathbf{t} \leq \begin{bmatrix}\mathbf{d}^{\mathrm{T}} & \mathbf{d}'^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$ *has the same solution set as* $\mathbf{At} \leq \mathbf{d}$, *then from Lemma 3.2,* $\begin{bmatrix}\widetilde{\mathbf{A}}^{\mathrm{T}} & \widetilde{\mathbf{A}}^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}} \mathbf{t} \leq \begin{bmatrix}\widetilde{\mathbf{d}}^{\mathrm{T}} & \widetilde{\mathbf{d}}'^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$ *has the same solution set as* $\widetilde{\mathbf{A}}\mathbf{t} \leq \widetilde{\mathbf{d}}$. *Assume to the contrary that there is some* $d^*_{i,j}$ *in* $\widetilde{\mathbf{d}}$ *and* $d'^*_{i,j}$ *in* $\widetilde{\mathbf{d}}'$ *such that* $d^*_{i,j} > d'^*_{i,j}$. *Since* $d^*_{i,j}$ *is the optimal solution to the linear program*

$$\begin{aligned} \textbf{maximize} \quad & t(e_i) - t(e_j) \\ \textbf{subject to} \quad & \widetilde{\mathbf{A}}\mathbf{t} \leq \widetilde{\mathbf{d}} \end{aligned} \tag{C.1}$$

and thus the optimal solution to the linear program (C.2)

$$\begin{aligned} \textbf{maximize} \quad & t(e_i) - t(e_j) \\ \textbf{subject to} \quad & \begin{bmatrix}\widetilde{\mathbf{A}} \\ \widetilde{\mathbf{A}}\end{bmatrix} \mathbf{t} < \begin{bmatrix}\widetilde{\mathbf{d}} \\ \widetilde{\mathbf{d}}'\end{bmatrix} \end{aligned} \tag{C.2}$$

However, the optimal solution to the linear program (C.2) can be at most $d'^*_{i,j}$ *when the solution set of* $\begin{bmatrix}\widetilde{\mathbf{A}}^{\mathrm{T}} & \widetilde{\mathbf{A}}^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}} \mathbf{t} \leq \begin{bmatrix}\widetilde{\mathbf{d}}^{\mathrm{T}} & \widetilde{\mathbf{d}}'^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$ *is not empty. The contradiction implies that* $\left\{\mathbf{t} \,\middle|\, \begin{bmatrix}\mathbf{A}^{\mathrm{T}} & \mathbf{A}'^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}} \mathbf{t} \leq \begin{bmatrix}\mathbf{d}^{\mathrm{T}} & \mathbf{d}'^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}\right\} = \{\mathbf{t}\,|\mathbf{At} \leq \mathbf{d}\} \Rightarrow \widetilde{\mathbf{d}} \leq \widetilde{\mathbf{d}}'$. □