Coordination Operators and their Composition under the Actor-Role-Coordinator (ARC) Model

Miao Song, Shangping Ren
Computer Science Department
Illinois Institute of Technology
Chicago, USA
{*msong8,ren*}*@iit.edu*

***Abstract*—For large open and distributed real-time applications, coordination constraints among concurrent, spatially distributed and autonomous entities can be complex. The Actor-Role-Coordinator (ARC) model we developed earlier [1] introduced the concept of *roles* which are abstractions of behaviors that are to be coordinated. Each role's behaviors may be shared by many concurrent entities, or played by many *actors*. Based on the role concept, coordination activities in large systems are partitioned into inter-role and intra-role coordinations to mitigate the coordination complexity. This paper focuses on coordination primitives and the composition of these primitives in forming more complex intra-role and inter-role coordination constraints. In particular, we define two primitive coordination operators, i.e., *precede* ($\preceq_t$) and *select* ($\trianglerighteq_p$), and use them to express temporal and spacial (with respect to actor system's behavioral space) coordination constraints among concurrent and autonomous actors. We further provide an operational semantics for these operators under the ARC model and provide case studies to illustrate their expressiveness in specifying complex coordination constraints.**

***Keywords*-Actor model; Actor-Role-Coordinator model; coordination; coordination constraints; coordination operators; transition systems; operational semantics**

I. Introduction

For large open distributed and real-time applications, there are often many concurrent asynchronous and distributed entities, and they communicate with each other through messages. Due to the dynamicity and openness of these applications, it is impetuous to assume synchrony among these entities. However, the virtue of real-time applications, on the other hand, requires certain temporal orders, or more generally, requires coordination constraints being enforced among these autonomous and concurrent entities.

Traditional approaches often embed these requirements inside implementation languages, operating systems, or system architectures, rather than treat coordination constraints as independent and first class entities at programming language level. One of the drawbacks of such approaches is that any modification of coordination constraints may require changes of the application's programming implementations, or the supporting run-time environment, such as operating systems, or system architectures.

The base of our model is build upon the belief that *how to implement a functionality and when/where to execute the functionality* are orthogonal. Computational entities implements the *how*, while *when and where to execute the functionality* is defined by coordination constraints which are encapsulated in coordination entities. As *how* and *when/where* are orthogonal, good programming models should ensure implementation transparency among the two orthogonal classes, i.e., among computational entities and coordination entities.

In the Actor-Role-Coordinator (ARC) model, the Actor model [2], [3] is used to model distributed and concurrent computation. Under the Actor model, each actor encapsulates a single thread of computation. It has states and communicates with other actors in the system by asynchronous messages. The Actor system only guarantees that a message sent by an actor will *eventually* be processed by its receiving actor. The only coordination constraints inherent in the Actor system is the *causal* order [4]. The ARC model adds a coordination layer (role and coordinator) on top of the actor system and externally enforces coordination constraints through manipulations of actor message dispatch time and dispatch location. As the Actor model is a pure asynchronous system model and does not assume any message delivery time, such message manipulations are transparent to actors and hence enable us to program computation (actors) and coordination (roles and coordinators) independently.

For large open distributed and real-time applications, coordination requirements among concurrent and autonomous entities can be complex. One of the challenging issues arises: is there a small set of primitive coordination operators through which or through their composition complex coordination requirements can be represented? This question is motivated by the existence of functionally completed logic operations. In logic domain, logical propositions can be complex, however, these complex propositions in fact can be represented by a small set of simple, but functionally completed logic operations, such as the set $\{\wedge, \neg\}$. In

this paper, we propose two coordination operators, i.e., *precede* ($\preceq_t$) which constraints the quantitative temporal order among two computational events, and *select* ($\trianglerighteq_p$) which constraints the actors in their behavioral space, and study their compositions.

The rest of paper is organized as follows: for self-containment, Section II briefly discusses the ARC model upon which our current work is built. Section III introduces the two primitive coordination operators and studies their properties and compositions. Section IV gives formal operational semantics for the two coordination operators under the ARC model. Section V discusses related work. We conclude in Section VI.

II. The Actor-Role-Coordinator (ARC) model

For self-containment, this section gives a brief overview of the ARC model, detailed discussion of the model and its implementation can be found in [1] and [5], respectively. As the model name indicates, there are three distinctive types of entities in the model, i.e., the actors, roles and coordinators. The actors encapsulate asynchronous, autonomous and concurrent computation, while the roles and coordinators encapsulate coordination constraints to be enforced upon the computation entities, i.e., the actors. The following subsections summarize the functionalities and properties of these three types of entities.

A. Actors

Actors in the ARC model are the same as the one defined in the Actor model [2], [3]. In particular, actors are single-threaded active objects. They communicate with each other *only* through asynchronous messages. Each actor has a mailbox where the received and yet to be processed messages are buffered. Actors have states and behaviors. The actor's current state and behavior decide how it processes messages dispatched on its active thread. The actor's state and behavior can only be changed by the actor itself while processing a message. The active threads within actors continuously process messages whenever their mailboxes are not empty. There are only three primitive operations each actor can perform, i.e., *create* new actors, *send* messages to other actors whom the sender knows the address of, and while processing a message, an actor can also perform *become* upon which the actor assumes a new state and a new behavior. All these operations are atomic. Therefore, we treat these operations as instantaneous events. The only order inherent in the Actor model is the causal order and the only guarantee provided by the model is that messages sent by actors will eventually be processed by their receiving actors. Other synchronization and coordination constraints for the purpose of quality of services hence need to be externally enforced. Roles and coordinators are to enforce such constraints.

B. Roles and Coordinators

Roles and coordinators in the ARC model constitute the coordination layer. Roles are introduced as behavior abstractions to mitigate coordination scalability and complexity issues inherent in large open distributed applications. With the introduction of roles, coordination is partitioned into two categories, i.e., inter-role coordination and intra-role coordination, which are the responsibility of coordinators and roles, respectively. Fig. 1 illustrates the layered structure of the ARC model.

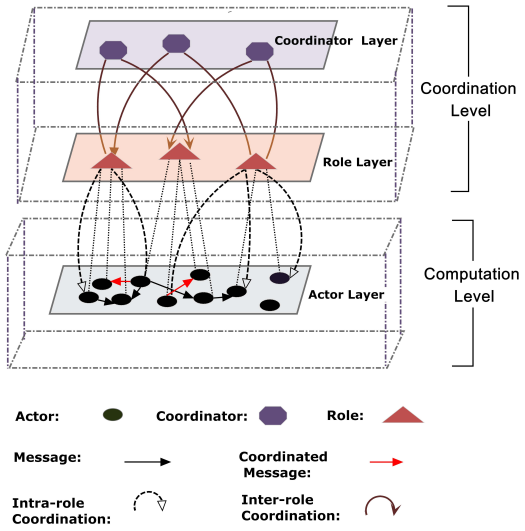

Figure 1. The Actor Role and Coordinator (ARC) Model

As shown in Fig. 1, the separation of concerns is apparent in the relationships among the layers. The actor layer is dedicated to computational behaviors and is oblivious to the coordination enacted in the role and coordinator layers. The roles and coordinators constitute the coordination layer responsible for imposing coordination constraints among actors. The coordinator layer is oblivious to the actor layer and is dedicated to inter-role coordination. The role layer bridges the actor layer and the coordinator layer and may therefore be viewed from two perspectives. From the perspective of a coordinator, a role enables coordination of a set of actors that share the static descriptions of an abstract behavior associated with the role without requiring the coordinator to have fine-grained knowledge of the individual actors that play the role. From the perspective of an actor, a role is an active coordinator that transparently manipulates the messages sent and received by the actor.

Roles and coordinators themselves can be viewed as meta actors and react to meta messages and actor events which are the occurrences of actors executing *create*, *send*, or *become*. The role meta actors are able to observe and manipulate messages in the actor layer. All events and message manipulations associated with an initial triggering

event or meta messages are indivisible and atomic, with no intermediate states visible across or within the three layers. Since the role and coordinator meta actors are state-based objects, the coordination policies within an application may adapt over time.

C. Inter-role and Intra-role Coordination constraints

Inter-role coordination constraints restrict abstract behaviors in temporal and actor behavioral space domains, while intra-role coordination constraints enforce the temporal and spacial (with respect to actor behavioral space) order upon actors which carry out the distributed computation.

We use a simplified robotic hand example to help understand the differences and relationships between inter-role and intra-role coordination constraints. Consider two robotic hands transferring an object from its left hand to its right hand and we assume each hand may have many fingers. If we use actors to model individual fingers, the left hand and the right hand are two different roles. The inter-role constraint for transferring an object from the left hand to the right hand is that the left hand must release the object before the right hand grasps the object; while the intra-role constraint for left hand is that all fingers must simultaneously release the object.

As we can see from this example, although the coordinatees of inter-role and intra-role coordinations are different, both constraints are to enforce certain orders among a selected set of actions.

The next section is to define two coordination operations and study their expressiveness under the ARC model.

III. Coordination Operations under the ARC Model

As we have mentioned in earlier sections, the underlying computational model that the ARC model built upon is an asynchronous model, i.e., the Actor model, which only guarantees the *causal order*. For applications with QoS requirements, such as real-time requirements, more restrictive orders and stronger guarantees are needed. In this section, we introduce two primitive coordination operators, i.e., *precede* ($\preceq_t$) which constraints the quantitative temporal order among two computational events, and *select* ($\trianglerighteq_p$) which constraints the actors in their behavioral space, and study their properties and compositions.

A. Terms and Notations

Before we present the two coordination operators, we first introduce a few terms and notations that are used throughout the paper.

1) Actor Behavior and Behavioral Space: Actor behaviors decide what type of messages an actor is able to process. An actor may have many different behaviors. However, at any given point in time, it can only manifest one behavior. For example, if we consider a wheel of a car as an actor, the wheel may "turn-right", "turn-left" or "stop". However, at any given time instance, it can only perform one and only one action. *Wheel:turn-right* defines the behavior of the wheel actor, i.e., the wheel actor processes the *turn-right* message. More specifically, Definition 1 formally defines the actor behavior.

Definitions 1 (Actor Behavior): An actor behavior is denoted as $[A :: M]$, where A is the actor's unique name representing the state of the actor, and M is an message instance the actor A processes.

□

The execution of the actor behavior $[A :: M]$ means that the message M is dispatched on the thread of actor A where it is processed. As message processing in the Actor model is atomic, we hence treat actor behaviors as instantaneous.

Definitions 2 (Actor Behavior Space): Given an actor system ($\mathbb{A}$), $\mathbb{M}$ is the message set that all the actors in $\mathbb{A}$ can process, the actor behavior space ($\mathbb{B}$) of the system $\mathbb{A}$ is define as

$$\mathbb{B} = \{[A :: M] \mid A \in \mathbb{A}, \text{and } M \in \mathbb{M}\} \cup \{\top, \bot\} \quad (1)$$

where $\top$ indicates the initial behavior when the system starts, and $\bot$ indicates when the system terminates.

□

Actor messages are processed in time. To obtain the time instances at which an actor executes a specific behavior, we define a time function $\mathcal{T}$ to project a specific actor behavior to the time domain.

Definitions 3 (Actor Behavior Time Function): The actor behavior time function $\mathcal{T} : \mathbb{B} \rightarrow \mathbb{R}_{\geq 0}$, where $\mathbb{B}$ is the actor behavior space of a given actor system, and $\mathbb{R}_{\geq 0}$ is non-negative real number set.

$$\mathcal{T}([A :: M]) = t \in \mathbb{R}_{\geq 0} \quad (2)$$

and $\mathcal{T}(\top) = 0$, $\mathcal{T}(\bot) = \inf$.

□

B. Temporal Constraint Operator: Precede

In the Actor model, as both actors and communications are asynchronous, no order assumptions are made among actors, implicit or explicit, other than that the *causal order* is obeyed by the actor system. In this subsection, we introduce a temporal coordination operator *proceed* which quantitatively constraints actor behaviors in the time domain. It is worth pointing out that we assume that all the actors share the same global wall-clock time.

Definitions 4 (Precede Operator: $\preceq_t$): The *precede* operator constrains quantitative temporal relationship between two behavior sets. $\{[A_{11} :: M_1 1], \cdots, [A_{1n} :: M_{1n}\}] \preceq_t \{[A_{21} :: M_{21}], \cdots, [A_{2m} :: M_{2m}]\}$ requires that $\max\{\mathcal{T}([A_{2j} :: M_{2j}])\} - \max\{\mathcal{T}([A_{1i} :: M_{1i}])\} = t$, where $\mathcal{T}$ is defined in Definition 3, $t \geq 0$, $1 \leq i \leq n$, and $1 \leq j \leq m$. As a syntactic sugar, we use $\mathcal{T}([A_2 ::$

$M_2]) - \mathcal{T}([A_1 :: M_1]) = t$, when the sizes of the behavior sets both equal to 1.

□

There are three special cases with respect to the value t associated with the constraint, i.e., when $t = 0$, t is not specified, or $t = \inf$. We discuss each in detail.

Case 1: when $t = 0$ ($\preceq_0$)

A behavior precedes another by a time quantity of zero amount indicates that the two involved behavior must happen simultaneously. It is worth pointing out that as actor systems are asynchronous, there is no guarantee that behaviors of different actors will happen at the "exact" same wall-clock time point. We use the word "simultaneously" to indicate that the happenings of the behaviors are atomic and no inter-medium states are visible. Hence the commonly encountered synchronization constraint becomes a special case of the quantitative precede constraint. As a syntactic sugar, we define *syn* operator.

Definitions 5 (Syn Operator):

$$\begin{aligned} &\text{syn}([A_1 :: M_1], [A_2 :: M_2]) \equiv \\ &[A_1 :: M_1] \preceq_0 [A_2 :: M_2] \end{aligned} \tag{3}$$

□

Case 2: when t is not specified ($\preceq$)

When the qualitative time amount is not specified, the *precede* constraint defines a precedence order between the involved two behaviors. For instance, $[A_1 :: M_1] \preceq [A_2 :: M_2]$ only restricts that $[A_1 :: M_1]$ happens *before* $[A_2 :: M_2]$.

Case 3: when $t = \inf$ ($\preceq_{\inf}$)

The semantics that a behavior must happen infinitely long before another behavior in fact disables the second behavior from happening. In other words, $[A_1 :: M_1] \preceq_{\inf} [A_2 :: M_2]$ means actor A_1's processing the message M_1 disables message M_2 being dispatched on actor A_2.

Based on the definition, the transitivity property of the *precede* operator becomes evident, i.e.,

Property 1 (Transitive Property):

$$\begin{aligned} &([A_1 :: M_1] \preceq_{t1} [A_2 :: M_2]) \wedge ([A_2 :: M_2] \preceq_{t2} [A_3 :: M_3]) \\ &\longrightarrow [A_1 :: M_1] \preceq_{t1+t2} [A_3 :: M_3] \end{aligned} \tag{4}$$

□

C. Behavioral Space Coordination Constraint Operator: Select

An actor behavior space as given in Definition 2 defines all the possible behaviors an actor system may have during the lifetime of its execution. We define a coordination operator *select* ($\trianglerighteq_p$) to describe a selection of participants in a coordinated group, where p is the selection criteria, represented by a logic expression.

As an example, consider a building instrumented with many smoke detectors at different locations and different floor levels. The requirement that if any of the smoke detectors has triggered an alarm, the fire door at the level where the alarm is located shall be closed in fact involves two selections, the selection of smoker detectors and the selection of fire doors.

Definitions 6 (Select Operator: $\trianglerighteq_p$): Given a set of behaviors in an actor system, $\mathbb{B} = \{\top, [A_1 :: M_1], \cdots, [A_i :: M_i], \cdots, [A_n :: M_n], \bot\}$, $\trianglerighteq_p(\mathbb{B})$ is function defined as $\trianglerighteq_p : 2^{\mathbb{B}} \longrightarrow \mathbb{B}$. $\trianglerighteq_p(\mathbb{B}) = [A_j :: M_j]$ indicates that $[A_j :: M_j] \in \mathbb{B}$, and $[A_j :: M_j]$ satisfies the selection criteria p. When $p = \texttt{true}$, it is omitted from the notation.

□

The definition of the *select* operator seems rather weak and does not provide much information other than that a behavior in a behavior group is selected to participate in a coordinated activities. In fact, the loosely defined *select* operator provides the intra-role coordinators, i.e., the roles, a powerful tool to select appropriate member actors to fulfill inter-role coordination without hard wiring to specific actors. The fire door and smoke detector scenario described above gives an example.

With the *select* operator, the coordination between smoke detectors and fire doors can be expressed as below:

$$\trianglerighteq_{p1}(\mathbb{SD}) \preceq_t \trianglerighteq_{p2}(\mathbb{FD}) \tag{5}$$

where, $\mathbb{SD}$ and $\mathbb{FD}$ are smoker detectors and fire doors behavior spaces, $p1$ and $p2$ are selection criteria for the detector and door, respectively.

As it can be seen, when we use door-role and detector-role to categorize two groups of actors, i.e., doors and detectors, respectively, the inter-role precedence constraint built upon the intra-role coordination operation *select* becomes rather static, and does not depend on which detector triggers an alarm.

The *select* operator selects only one behavior from a given group, multiple selections can be done through iteratively use of the *select* operator. For instance, formula 6 non-deterministically selects two elements from a given behavior group $\mathbb{B}$:

$$\{\trianglerighteq_p(\mathbb{B})\} \cup \{\trianglerighteq_p(\mathbb{B} - \{\trianglerighteq_p(\mathbb{B})\})\} \tag{6}$$

where $-$ represents set subtractions.

Similar to introducing the *syn* operator as a syntactic sugar for convenience purpose, we introduce multiple selection operator $\trianglerighteq_p^n$.

Definitions 7 (Multiple Selection Operator: $\trianglerighteq_p^n$):

$$\trianglerighteq_p^n(\mathbb{B}) = \{b_i \mid b_i \in \mathbb{B}, 0 \leq i \leq n, n \leq |\mathbb{B}|, p(b_i) = \text{True}\}$$

where $|\mathbb{B}|$ is the cardinality of the behavior set $\mathbb{B}$. When all behaviors that satisfy p are chosen, we use *all* in the place of n. Furthermore, when p = `True`, we omit it from the notation.

□

In the smoke detector example, if we would like to prevent false alarm caused by smoke detector malfunction, and require two smoke detectors simultaneously set on alarm before a fire door is closed, we can easily express the coordination requirement as shown below:

$$syn(\trianglerighteq^2_{p1}(\mathbb{SD})) \preceq_t \trianglerighteq_{p2}(\mathbb{FD}) \tag{7}$$

Before we complete this section, we give a few examples to illustrate the use of these operators, and show their expressiveness.

D. Examples

In this subsection, we use a lighting system as an example to illustrate the use of proposed coordination operators.

> **Example:** Consider a building with three rooms, *room1*, *room2* and *room3*. *Room1* and *room2* have two lights, a_1 and a_2 in *room1*, and a_3 and a_4 in *room2*, while *room3* has only one light a_5. All the lights work independently and they can be turned on, off, or glittering. Furthermore, each light has a property to indicate if it is of an energy-saving type.

Assuming all the lights are off when the lighting system starts, we study the following coordination scenarios:

1) All lights in *room1*, a_1 and a_2, glitter simultaneously.
2) The light a_5 in *room3* glitters every 5 time units, and glitters for 3 times.
3) The lights in all three rooms with energy-saving property must be turned on within 5 time units, but the order in which they are turned on can be arbitrary.
4) In each room, all lights must be turned on within 5 time units. However, if there are both energy-saving and non-energy-saving lights, the energy-saving lights must be turned on before the non-energy-saving one.
5) There are two light glitters in *room2*. We require that the two glitters come from two different lights.
6) There are two light glitters in *room2*. We require that the two glitters come from the same light, but we do not care which light the glitters come from.
7) The two lights in *room1* may take action of turning on or glittering, we require that 2 time unit after the *room1*'s two lights take their actions, a light in *room2* and a light in *room3* must be simultaneously turned on.

If we model the lights as actors, these actors accept three types of messages, i.e., *turn-on* (M_{on}), *turn-off* (M_{off}), and *glitter* (M_{glr}). The actor behavior space of the lighting system in the building can hence be given by the following:

$$\begin{aligned}\mathbb{B} =&\{[a_1 :: M_{on}], [a_1 :: M_{off}], [a_1 :: M_{glr}], \cdots, \\ &[a_5 :: M_{on}], [a_5 :: M_{off}], [a_5 :: M_{glr}]\}\end{aligned}$$

The lighting system's behavior space, $\mathbb{B}$, can be further partitioned into different small actor behavior spaces based on different criteria, or role abstraction. If we take the location of actors into consideration, we have behavior space for each room, i.e., $\mathbb{B}_{r1}$, $\mathbb{B}_{r2}$ and $\mathbb{B}_{r3}$, where

$$\begin{aligned}\mathbb{B}_{r1} =&\{[a_1 :: M_{on}], [a_1 :: M_{off}], [a_1 :: M_{glr}], [a_2 :: M_{on}], \\ &[a_2 :: M_2], [a_2 :: M_{glr}]\}, \\ \mathbb{B}_{r2} =&\{[a_3 :: M_{on}], [a_3 :: M_{off}], [a_3 :: M_{glr}], [a_4 :: M_{on}], \\ &[a_4 :: M_4], [a_4 :: M_{glr}]\}, \\ \mathbb{B}_{r3} =&\{[a_5 :: M_{on}], [a_5 :: M_{off}], [a_5 :: M_{glr}]\}\end{aligned}$$

Clearly, if we partition the behavior space based on a different abstraction, such as on the type of messages, we have different subset of behavior space as following:

$$\begin{aligned}&\mathbb{B}_{M_{on}} = \{[a_1 :: M_{on}], [a_2 :: M_{on}], \cdots, [a_5 :: M_{on}]\}, \\ &\mathbb{B}_{M_{off}} = \{[a_1 :: M_{off}], [a_2 :: M_{off}], \cdots, [a_5 :: M_{off}]\}, \\ &\mathbb{B}_{M_{glr}} = \{[a_3 :: M_{glr}], [a2 :: M_{glr}], \cdots, [a_5 :: M_{glr}]\}\end{aligned}$$

The role in the ARC model in fact defines these individual behavior spaces. These subspaces are *partitions* of the system's behavior space. In particular, we have

$$\begin{aligned}&\mathbb{B}_{r_i} \cap \mathbb{B}_{r_j} = \phi, \ \textit{if } i \neq j \\ &\mathbb{B}_{M_i} \cap \mathbb{B}_{M_j} = \phi, \ \textit{if } i \neq j \\ &\mathbb{B} = \mathbb{B}_{r_1} \cup \mathbb{B}_{r_2} \cup \mathbb{B}_{r_3} \\ &\mathbb{B} = \mathbb{B}_{M_{on}} \cup \mathbb{B}_{M_{off}} \cup \mathbb{B}_{M_{glr}}\end{aligned}$$

1) All lights in *room1*, i.e., a_1 and a_2, glitter simultaneously.

$$syn(\{[a_1 :: M_{glr}], [a_2 :: M_{glr}]\}) \tag{8}$$

or, it can also be represented by

$$syn(\trianglerighteq^{all}(\mathbb{B}_{r1} \cap \mathbb{B}_{M_{glr}})) \tag{9}$$

The representation of (8) directly binds two specific actors. Hence, if the number of lights in the room changes, the constraint specified will have to make corresponding change. The representation of (9) (which is based on the role, or behavioral space) is, on the other hand, oblivious to such changes.

2) The light a_5 in *room3* glitters every 5 time units, and glitters for 3 times.

$$\begin{aligned}&[a_5 :: M_{glr,1}] \preceq_5 [a_5 :: M_{glr,2}] \preceq_5 [a_5 :: M_{glr,3}] \\ &\preceq_{\inf} \trianglerighteq(\mathbb{B}_{r3} \cap \mathbb{B}_{M_{glr}})\end{aligned} \tag{10}$$

The coordination constraint (10) disables any glittering in *room3* after the light has glittered for 3 times.

3) The lights in all three rooms with energy-saving property must be turned on within 5 time units, but the order in which they are turned on can be arbitrary.

$$\top \preceq_5 \trianglerighteq^{all}_p(\mathbb{B}_{M_{on}}) \tag{11}$$

where p is the criteria for *energy saving* property.

4) In each room, all lights must be turned on within 5 time units. However, if there are both energy-saving and non-energy-saving lights, the energy-saving lights must be turned on before the non-energy-saving one.

$$\begin{aligned}&\top \preceq_5 \trianglerighteq^{all}_{\texttt{true}}(\mathbb{B}_{M_{on}})\wedge\\&\trianglerighteq^{all}_{p}(\mathbb{B}_{M_{on}}) \preceq \trianglerighteq^{all}_{\neg p}(\mathbb{B}_{M_{on}})\end{aligned} \tag{12}$$

where p is the criteria for *energy saving* property.

5) There are two light glitters in *room2*. We require that the two glitters come from two different lights.

$$\trianglerighteq(\mathbb{B}_{r2} \cap \mathbb{B}_{M_3}) \preceq \trianglerighteq(\mathbb{B}_{r2} \cap \mathbb{B}_{M_3} - \{b\}) \tag{13}$$

where $b = \trianglerighteq(\mathbb{B}_{r2} \cap \mathbb{B}_{M_3})$

6) There are two light glitters in *room2*. We require that the two glitters come from the same light, but we do not care which light the glitters come from.
We define a function $\mathcal{F} : \mathbb{B} \rightarrow \mathbb{A}$ to extract corresponding actor(s) for a given actor behavior, and function $\mathcal{I} : \mathbb{A} \rightarrow \mathbb{N}$ to extract the unique identifier for a given actor. With the two supporting functions, the coordination requirement can be represented as (14).

$$\trianglerighteq(\mathbb{B}_{r2} \cap \mathbb{B}_{M_{glr}}) \preceq \trianglerighteq_p(\mathbb{B}_{r2} \cap \mathbb{B}_{M_{glr}}) \tag{14}$$

where is the property given below:

$$p : \mathcal{I}(\mathcal{F}(\mathbb{B}_{r2} \cap \mathbb{B}_{M_{glr}})) = \mathcal{I}(\mathcal{F}(\trianglerighteq(\mathbb{B}_{r2} \cap \mathbb{B}_{M_{glr}}))$$

The predicate p requires that the actor identifier from the second select must be the same as the one chosen from the first select.

7) The two lights in *room1* may take action of turning on or glittering, we require that 2 time unit after the *room1*'s two lights take their actions, a light in *room2* and a light in *room3* must be simultaneously turned on.

$$\begin{aligned}&\trianglerighteq^{2}(\mathbb{B}_{r1} \cap (\mathbb{B}_{M_{on}} \cup \mathbb{B}_{M_{glr}}))\\&\preceq_2 \mathit{syn}\{\trianglerighteq(\mathbb{B}_{r2} \cap \mathbb{B}_{M_{on}}), \trianglerighteq(\mathbb{B}_{r3} \cap \mathbb{B}_{M_{on}})\}\end{aligned} \tag{15}$$

As we can see from these exercises that complex coordination constraints can be concisely represented by the two proposed coordination operators, i.e., *precede* and *select*. Furthermore, the concept of roles, or behavior spaces, enables more static and flexible constraint representations that are resilient to changes at the lower computation layer.

Next section, we provide formal semantics for the two operators.

IV. Operational Semantics of *Precede* and *Select* under the ARC Model

Before we give the operational semantics for the proposed two coordination operators, we first introduce configurations that are used to define an ARC system's operational semantics. A configuration captures the system's state and the system's operational semantics is given by the configuration transitions. The notations are adopted from the Actor model.

Definitions 8 (Actor Configuration):

An actor configuration contains an actor map α, and multi-set of messages, μ. It is represented as:

$$\langle\langle \alpha \mid \mu \rangle\rangle \tag{16}$$

□

where the actor map α maps an actor's unique identifier to its current state and corresponding behavior.

Definitions 9 (Role Configuration):

A role configuration contains a set of actors playing the role, α_γ , the role itself, γ, and a multi-set of messages stored in the mailboxes of the actors playing the role, μ_γ. It is denoted as:

$$\langle\langle \alpha_\gamma, \gamma \mid \mu_\gamma \rangle\rangle \tag{17}$$

□

A. Operational Semantics for Precede ($\preceq_t$)

Based on the specification of t, we have four different system transitions as below.

Case 1: when two behaviors must be executed simultaneously, i.e, $t = 0$ ($\preceq_0$).

$$\begin{aligned}&\langle\langle \alpha^1, \cdots, \alpha^i_{\gamma_1}, \cdots, \alpha^j_{\gamma_2}, \cdots, \alpha^n, \gamma_1, \gamma_2, \cdots, \gamma_m \mid \mu, m^i_{\gamma_1}, m^j_{\gamma_2} \rangle\rangle\\&\xrightarrow{[\alpha^i_{\gamma_1} :: m^i_{\gamma_1}] \preceq_0 [\alpha^j_{\gamma_2} :: m^j_{\gamma_2}]}\\&\langle\langle \alpha^1, \cdots, \alpha^i_{\gamma_1}[m^i_{\gamma_1}], \cdots, \alpha^j_{\gamma_2}[m^j_{\gamma_2}], \cdots, \alpha^n, \gamma_1, \gamma_2, \cdots, \gamma_m \mid \mu \rangle\rangle\end{aligned} \tag{18}$$

Case 2: when the second behavior is disabled, i.e., $t = \inf$ ($\preceq_{\inf}$).

$$\begin{aligned}&\langle\langle \alpha^1, \cdots, \alpha^i_{\gamma_1}, \cdots, \alpha^j_{\gamma_2}, \cdots, \alpha^n, \gamma_1, \gamma_2, \cdots, \gamma_m \mid \mu, m^i_{\gamma_1}, m^j_{\gamma_2} \rangle\rangle\\&\xrightarrow{[\alpha^i_{\gamma_1} :: m^i_{\gamma_1}] \preceq_{\inf} [\alpha^j_{\gamma_2} :: m^j_{\gamma_2}]}\\&\langle\langle \alpha^1, \cdots, \alpha^i_{\gamma_1}[m^i_{\gamma_1}], \cdots, \alpha^j_{\gamma_2}, \cdots, \alpha^n, \gamma_1, \gamma_2, \cdots, \gamma_m \mid \mu \rangle\rangle\end{aligned} \tag{19}$$

Case 3: when only qualitative temporal precedence is required, i.e., t is not specified ($\preceq$).

In this case, the intermediate state becomes visible to the outside world while the first behavior is executed. In other words, from system configuration perspective, we will see one message is dispatched on its destination actor, while the other may still remain in its mail-queue.

$$\begin{aligned}&\langle\langle \alpha^1, \cdots, \alpha^i_{\gamma_1}, \cdots, \alpha^j_{\gamma_2}, \cdots, \alpha^n, \gamma_1, \gamma_2, \cdots, \gamma_m \mid \mu, m^i_{\gamma_1}, m^j_{\gamma_2} \rangle\rangle\\&\xrightarrow{[\alpha^i_{\gamma_1} :: m^i_{\gamma_1}] \preceq [\alpha^j_{\gamma_2} :: m^j_{\gamma_2}]}\\&\langle\langle \alpha^1, \cdots, \alpha^i_{\gamma_1}[m^i_{\gamma_1}], \cdots, \alpha^j_{\gamma_2}, \cdots, \alpha^n, \gamma_1, \gamma_2, \cdots, \gamma_m \mid \mu, m^j_{\gamma_2} \rangle\rangle\end{aligned} \tag{20}$$

Case 4: when quantitative temporal precedence is required, i.e., $0 \leq t \leq \inf\ (\preceq_t)$.

This is a more complicated case in which a timer is needed. When the precedent behavior is executed, it sets the timer for the amount required. When the timer reaches 0, it immediately triggers the second behavior. Therefore, two system configuration transitions are involved as given in (21) and (22).

$$\langle\langle \alpha^1, \cdots, \alpha^i_{\gamma_1}, \cdots, \alpha^j_{\gamma_2}, \cdots, \alpha^n, \gamma_1, \cdots, \gamma_m, \tau \mid \mu, m^i_{\gamma_1}, m^j_{\gamma_2} \rangle\rangle$$

$$\xrightarrow{[\alpha^i_{\gamma_1} :: m^i_{\gamma_1}] \preceq_t [\alpha^j_{\gamma_2} :: m^j_{\gamma_2}]}$$

$$\langle\langle \alpha^1, \cdots, \alpha^i_{\gamma_1}[m^i_{\gamma_1}], \cdots, \alpha^j_{\gamma_2}, \cdots, \alpha^n, \gamma_1, \cdots, \gamma_m, \tau[t] \mid \mu, m^j_{\gamma_2} \rangle\rangle \tag{21}$$

$$\langle\langle \alpha^1, \cdots, \alpha^i_{\gamma_1}, \cdots, \alpha^j_{\gamma_2}, \cdots, \alpha^n, \gamma_1, \gamma_2, \cdots, \gamma_m, \tau[t] \mid \mu, m^j_{\gamma_2} \rangle\rangle$$

$$\xrightarrow{syn(\tau[t] = 0, [\alpha^j_{\gamma_2} :: m^j_{\gamma_2}])}$$

$$\langle\langle \alpha^1, \cdots, \alpha^i_{\gamma_1}, \cdots, \alpha^j_{\gamma_2}[m^j_{\gamma_2}], \cdots, \alpha^n, \gamma_1, \gamma_2, \cdots, \gamma_m, \tau \mid \mu \rangle\rangle \tag{22}$$

B. Operational Semantics for Select ($\trianglerighteq_p$)

The operational semantics of the *select* ($\trianglerighteq_p$) operator performed by the roles is given the following interpretation. The selection criteria of the $\trianglerighteq_p$ operator, p, is a role state dependent propositional function, and its selection space is the behavior space of the role's member actors . For a given role γ, if $[A :: M] \in \mathbb{B}_\gamma$, $p(\gamma, [A :: M]) = \texttt{True}$, and $[A :: M]$ is selected, i.e., $\trianglerighteq_p(\mathbb{B}_\gamma) = [A :: M]$, then the behavior $[A :: M]$ is either constrained by a inter-role coordination constraint imposed by a coordinator, or the execution of the behavior disables the rest of the behaviors that satisfy the same selection criteria p. Formula (23) gives its operational semantics.

$$\langle\langle \alpha^1_\gamma, \cdots, \alpha^i_\gamma, \cdots, \alpha^j_\gamma, \cdots, \alpha^n_\gamma, \gamma[\trianglerighteq_p] \mid \mu_\gamma, m^i_\gamma, \cdots, m^j_\gamma \rangle\rangle$$

$$\xrightarrow{p_{i \leq l \leq j}(\gamma, \alpha^l_\gamma, m^l_\gamma) = \texttt{true}}$$

$$\left\langle\left\langle \alpha^1_\gamma, \cdots, \alpha^i_\gamma, \cdots, \alpha^l_\gamma \cdots, \alpha^j_\gamma, \cdots, \alpha^n_\gamma, \gamma \mid \mu_\gamma \right\rangle\right\rangle \tag{23}$$

Disabling a behavior is done by permanently removing the message without dispatching the message to its destination actor.

As we can see from the formal definition of *select*, the selection from qualified message dispatching is not predetermined, and it is rather arbitrary. The decision is to minimize unnecessary hard-wires and maximize the flexibility to accommodate the dynamic nature of the open systems.

V. Related Work

Coordination models can be categorized into two classes, i.e., control-driven (also called dataflow-driven) and data-driven. A broad survey on coordination models and languages can be found in [6], [7]. In data-driven models such as Linda and its extensions [8], coordination tends to be endogenous and embedded within computational entities. In control-driven models, coordination tends to be exogenous and isolated from computational entities. ABT [9], ROAD [10], IWIM, and CoLaS [11] are examples of control-driven coordination models. Hybrid approaches such as tuple centres and ReSpecT [12] combine the data-driven and control-driven models.

Reo [13], [14], [15], [16] is built on IWIM and ABT models. The abstract communication medium is a channel with exactly two ends and a constraint that relates the flow of data at its ends. Channels are connected to make a circuit by joining channel ends together to form nodes. The coordination in Reo is abstracted as a Reo circuit specified by a constraint automaton [13], while in the PBRD [17] model, coordinations are described as informal rule specifications on the resulting interactions of coordinated actors. A detailed comparison among the ARC, Reo and PBRD model can be found in [18].

Some control-driven models, such as ROAD, CoLaS, and Finesse [19], target the scalability issues of open distributed systems through group-based coordination models. Most current role-based coordination models are based on organizational concepts, where roles abstract coordination behaviors among participants that play the roles. Role-based coordination models are surveyed in [20].

Many recent researches have focused on application specific coordination problems. For instance, Fok et.al. [21] proposed a new service provisioning based middle-ware to collaborate heterogeneous devices in wireless sensor networks. Similarly, in order for software developer to adjust to ad hoc, heterogeneous, and changing hardware architecture, Bouhadiba et.al. [22] proposed the notion of "contract" and associated it with a component-based description framework to describe complex hardware behaviors. An event-based coordination model [23] extends the CEAOP by modeling coordination of concurrent adaptation rules as explicit contexts to be applied in context-aware applications. A coordination model to manage collaborative real-time editing work is proposed in [24]. A visual dataflow language tailored towards mobile applications is presented [25] as a separate coordination language to express the interactions among mobile components that operate on data streams.

The focus of this paper is to define a small set of coordination operators that are capable of expressing complex coordination constraints among autonomous, concurrent and asynchronous entities in distributed and open systems.

VI. Conclusion and Future work

In this paper, we have proposed two basic coordination operators, i.e., *precede* ($\preceq_t$) and *select* ($\trianglerighteq_p$) and further used examples to illustrate the expressiveness of these operators in forming more complex coordination constraints. The *precede* operator in essence controls the scheduling order of message dispatches among actors; and the *select* operator corresponds to a decision control over which actor a message is dispatched to. We feel these two coordination operations together with first order logic operations are functionally complete with respect to expressing coordination constraints. However, the formal proof of its completeness, or finding a counter-example that indicates its insufficiency, is yet to be done and will be our future research focus.

Acknowledgment

The research is supported by NSF CAREER Award (CNS 0746643) and CNS 1035894.

References

[1] S. Ren, Y. Yu, N. Chen, K. Marth, P. Poirot, and L. Shen, "Actors, Roles and CoordinatorsA Coordination Model for Open Distributed and Embedded Systems," in *Coordination Models and Languages*. Springer, 2006, pp. 247–265.

[2] G. Agha, "Actors: a model of concurrent computation in distributed systems," *AITR-844*, 1985.

[3] G. Agha, P. Thati, and R. Ziaei, "Actors: a model for reasoning about open distributed systems," in *Formal methods for distributed processing*. Cambridge University Press, 2001, p. 176.

[4] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[5] N. Chen and S. Ren, "Building a coordination framework to support behavior-based adaptive checkpointing for open distributed embedded systems," in *Proceedings of the 40th Annual Hawaii International Conference on System Science*, 2007.

[6] G. Papadopoulos and F. Arbab, "Coordination models and languages," *Advances in Computers*, vol. 46, pp. 329–400, 1998.

[7] F. Arbab, "Composition of Interacting Computations," *Interactive Computation*, pp. 277–321, 2006.

[8] G. Picco, A. Murphy, and G. Roman, "LIME: Linda meets mobility," in *Proceedings of the 21st international conference on Software engineering*. ACM, 1999, pp. 368–377.

[9] F. Arbab, "Abstract behavior types: A foundation model for components and their composition," in *Formal Methods for Components and Objects*. Springer, 2003, pp. 33–70.

[10] A. Colman and J. Han, "Coordination systems in role-based adaptive software," in *Coordination Models and Languages*. Springer, 2005, pp. 63–78.

[11] J. Cruz and S. Ducasse, "A group based approach for coordinating active objects," *Coordinatio Languages and Models*, pp. 15–15, 1999.

[12] A. Omicini, "Formal ReSpecT in the A&A perspective," *Electronic Notes in Theoretical Computer Science*, vol. 175, no. 2, pp. 97–117, 2007.

[13] C. Baier, M. Sirjani, F. Arbab, and J. Rutten, "Modeling component connectors in Reo by constraint automata," *Science of Computer Programming*, vol. 61, no. 2, pp. 75–113, 2006.

[14] S. Tasharofi and M. Sirjani, "Formal modeling and conformance validation for WS-CDL using Reo and CASM," *Electronic Notes in Theoretical Computer Science*, vol. 229, no. 2, pp. 155–174, 2009.

[15] F. Arbab, L. Aştefănoaei, F. de Boer, M. Dastani, J. Meyer, and N. Tinnermeier, "Reo connectors as coordination artifacts in 2APL systems," *Intelligent Agents and Multi-Agent Systems*, pp. 42–53, 2008.

[16] N. Kokash and F. Arbab, "Applying Reo to service coordination in long-running business transactions," in *Proceedings of the 2009 ACM symposium on Applied Computing*. ACM, 2009, pp. 1381–1382.

[17] C. Talcott, "Policy-based coordination in pagoda: A case study," *Electronic Notes in Theoretical Computer Science*, vol. 181, pp. 97–112, 2007.

[18] C. Talcott, M. Sirjani, and S. Ren, "Comparing three coordination models: Reo, ARC, and PBRD," *Science of Computer Programming*, 2009.

[19] A. Berry and S. Kaplan, "Open, distributed coordination with finesse," in *Proceedings of the 1998 ACM symposium on Applied Computing*. ACM, 1998, p. 184.

[20] G. Cabri, L. Ferrari, and F. Zambonelli, "Role-based approaches for engineering interactions in large-scale multi-agent systems," *Software Engineering for Multi-Agent Systems II*, pp. 360–361, 2004.

[21] C. Fok, G. Roman, and C. Lu, "Enhanced coordination in sensor networks through flexible service provisioning," in *Coordination Models and Languages*. Springer, 2009, pp. 66–85.

[22] T. Bouhadiba and F. Maraninchi, "Contract-based coordination of hardware components for the development of embedded software," in *Coordination Models and Languages*. Springer, 2009, pp. 204–224.

[23] A. Núñez and J. Noyé, "An event-based coordination model for context-aware applications," in *Proceedings of the 10th international conference on Coordination models and languages*. Springer-Verlag, 2008, pp. 232–248.

[24] A. Imine, "Coordination model for real-time collaborative editors," in *Coordination Models and Languages*. Springer, 2009, pp. 225–246.

[25] A. Lombide Carreton and T. DHondt, "A Hybrid Visual Dataflow Language for Coordination in Mobile Ad Hoc Networks," in *Coordination Models and Languages*. Springer, 2010, pp. 76–91.