Comparing Three Coordination Models: Reo, ARC, and RRD

Carolyn Talcott[1]

SRI International
Menlo Park, CA 94025, USA

Marjan Sirjani[2]

University of Tehran and IPM
Tehran, Iran

Shangping Ren[3]

Illinois Institute of Technology
Chicago, IL 60616, USA

Abstract

Three models of coordination—Reo, Actors-Roles-Coordinators (ARC), and Reflective Russian Dolls (RRD)—are compared and contrasted according to a set of coordination features. Mappings between their semantic models are defined. Use of the models is illustrated by a small case study.

Keywords: coordination model, semantics, Reo, Actor, Role, Reflective Russian Dolls

1 Introduction

Coordination is becoming an increasingly important paradigm for systems design and implementation. With multiple languages and models for coordination emerging it is interesting to compare different models and understand their strengths and weaknesses, find common semantic models and develop mappings between formalisms. This will help us to gain a deeper insight into coordination concepts and applications, and also to establish a set of features/criteria for defining and comparing coordination models. In this paper, we compare and contrast three coordination models: Reo [4], Actors-Roles-Coordinators (ARC) [20], and Reflective Russian Dolls (RRD) [17]. These three models cover a wide spectrum

[1] Email: clt@cs.stanford.edu

[2] Email: Marjan.Sirjani@cwi.nl

[3] Email: ren@iit.edu. This work is in part supported by NSF under grant CNS 0431832.

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

of communication mechanisms and coordination strategies and serve as a good sample set for our study. Other models to consider in future comparison studies include: Linda [13] and its mobile extension, Lime [19], Klaim[18] and its stochastic extension [11].

Reo is a channel-based exogenous coordination model for component composition. In Reo, complex connectors are compositionally built out of simpler ones. The simplest connectors are channels with well-defined behaviors. These connectors are represented graphically as circuits. Similar to electronic circuits, connectors show how distributed coordinatees are connected [4]. 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 coordinatees. Compositional semantics of Reo circuits can be given by Timed Data Streams (TDS) [8] and by constraint automata [9]. Constraint automata can also be used for analyzing and model checking Reo systems.

ARC uses the separation of concern principle to partition coordination into two disjoint categories, i.e., intra-role and inter-role coordination, and uses roles and coordinators, respectively, to abstract these behaviors. The coordinatees in the ARC model are actors, entities that interact by asynchronous message exchange. Coordination is through message time-space manipulations which are transparent to the coordinatees. Reasoning in the ARC system is based on message dispatches in time (when) and space (to whom).

RRD is a model of reflective distributed object computation. It uses reflection and hierarchical structure to provide a general layered coordination model. Each layer (meta-object) controls the communication and the execution of objects in the layer below. Policy-based RRD (PBRD) is a restricted form of RRD in which communication control is specified by declarative policies. As for ARC the objects being coordinated are actor-like objects. The semantics of RRD coordinators and coordinatees is interaction semantics [23,12] which is compositional both horizontally (composing object or coordinated object configurations) and vertically (composing coordinators and coordinatees).

The remainder of the paper is organized as follows. In Section 2, we spell out the features to be compared and contrasted. Section 3 describes the three models and compares them according to the listed features. Section 4 describes representations in the three models of a simple coordination task. In Section 5 we make a step towards a common semantic foundation for the three models. Conclusions and future work are discussed in Section 6.

2 Coordination Features

Coordination languages and models are being developed to address the problem of managing the interactions among concurrent and distributed processes. The underlying principle is separation of computations by components and their interactions [14,3]. In our study of the three chosen models of coordination we considered a number of features (dimensions in the design space) including those summarized below.

Computation model. Is communication message-, event-, or channel-based? Is it synchronous or asynchronous? Is state localized or is there a shared global memory? Is the state space discrete, continuous, or hybrid?

Control. Is the coordinator in control or is it a passive information store (control oriented versus data oriented coordination)? Do the coordinated components have explicit actions

[4] We use the term *coordinatee* through the text by which we mean the entities being coordinated.

for effecting the coordination?

Semantic model. How is the semantics of components and/or coordinators specified? An operational semantics could be given as a state transition system, such as automata or rewrite systems. Denotational semantics might be expressed in terms of observable events, traces/streams, or signals.

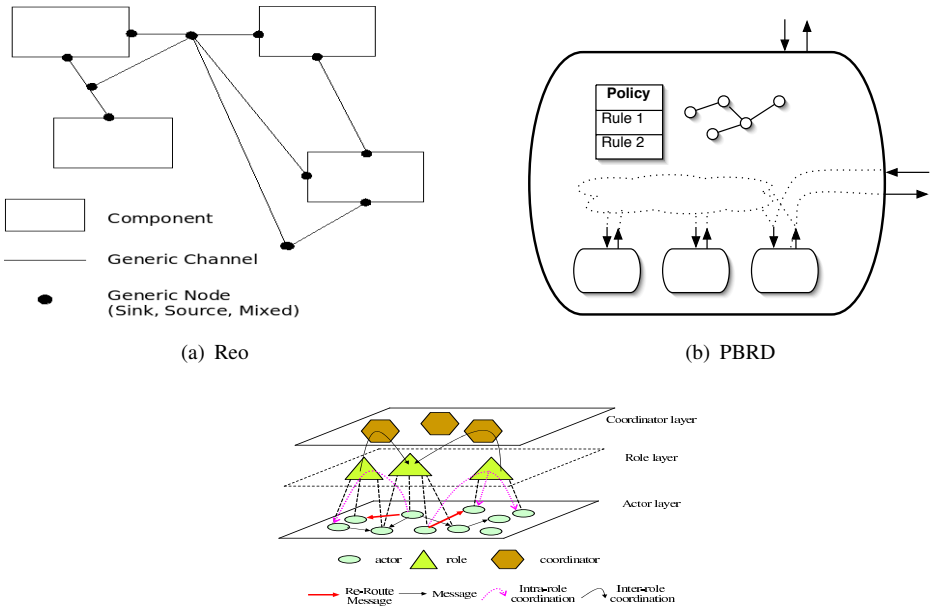
Modularity and Compositionality. An important issue is compositionality of system descriptions and semantics at all levels, both vertically and horizontally. Does the model provide mechanisms for structuring or modularizing coordination activities?

Specification. Coordination models typically focus on how a coordinator achieves its goals. But how are the goals specified? How can you decide if a coordinator achieves its goals? Examples of different kinds of goals include: serializing requests to a component; ensuring a given group communication semantics; ensuring atomicity of a group of messages; providing fault tolerance; and balancing resource usage, quality and timeliness.

Analyzability. An important and often ignored aspect of specifications is analyzability. To what degree do different coordination models support analyzability, verification of certain properties? And how?

3 Three Models of Coordination

Figure 1 gives a graphical impression of the Reo, ARC, and (policy-based) RRD. In the following each model is described in more detail, then we give a feature-wise comparison.



(a) Reo

(b) PBRD

(c) ARC

Fig. 1. Three Different Coordination Models

3.1 Reo

Reo is an exogenous coordination language based on a calculus of channel composition. A channel is an abstract communication medium with exactly two ends and a constraint that relates the flow of data at its ends. A channel represents a primitive interaction (protocol), explicitly represented as a binary constraint. There are two types of channel ends, source-end where data enters into the channel, and sink-end where data leaves the channel. A channel can have two sources, two sinks, or a source and a sink. The channel relation can be defined by users which allows an open-ended set of different channel types, each with its own policy for synchronization, buffering, ordering, computation, data retention/loss, etc.

Channels are connected to make a circuit by *joining* channel ends together to form *nodes*. A node is a *source node* if all of its channel ends are source ends. It is a *sink node* if channel ends are sink ends. Otherwise it is a *mixed node*. A component can write data items to a source node that it is connected to. The write operation succeeds only if all (source) channel ends coincident on the node accept the data item, in which case the data item is written to every source end coincident on the node. A source node, thus, acts as a *replicator*. A component can obtain data items, by a *take* operation, from a sink node that it is connected to. A *take* operation succeeds only if at least one of the (sink) channel ends coincident on the node offers a suitable data item; if more than one coincident channel end offers suitable data items, one is selected nondeterministically. A sink node, thus, acts as a nondeterministic *merger*. A mixed node nondeterministically selects and takes a suitable data item offered by one of its coincident sink channel ends and replicates it into all of its coincident source channel ends. The source or sink nodes which are the interfaces of a component and its environment are called (input or output) ports. Mixed nodes cannot be used as ports and are not available for other components to connect to. Assuming a Reo connector as a component, we may talk about ports of a connector.

Constraint automata are proposed in [7,9] as a compositional semantics of Reo. The automata-states stand for the possible configurations (e.g., the contents of the FIFO-channels of a Reo-connector) while the automata-transitions represent the possible data flow and its effect on these configurations. For each transition of a constraint automata there is a set of names which are fired and an expression showing the data constraint of that transition.

3.2 Actor-Role-Coordinator (ARC) Model

The main design goal of the Actor-Role-Coordinator (ARC) model is to facilitate open distributed embedded (ODE) system design and development. The intrinsic properties of ODE systems are: large scale, dynamic configuration, and limited resources but stringent multi-dimensional Quality of Service (QoS) requirements. Hence, beyond traditional synchronization of functional activities among large scale and dynamic embedded entities, the ARC model also provides a way to coordinate non-functional behaviors when different QoS requirements may not be concurrently satisfiable.

ARC is a role-based coordination model where a role is a static abstraction for a set of behaviors that underlying actors share. The dynamic aspect of the role is to coordinate its members. This type of coordination is called intra-role coordination. The intra-role coordination is achieved through message rerouting and reordering among actors within

the role. The coordination among different roles, i.e., inter-role coordination, on the other hand, is done by coordinators. Coordinators constrain roles' coordination behavior which eventually affects message dispatch time and location. However, actors and coordinators are transparent to each other. Hence, the dynamicity inherent in the embedded entities are hidden from the coordinators.

From the coordinatee 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. Conceptually, events are broadcast and the system guarantees event delivery atomicity among all entities interested in the events. 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 invariants are a composition of intra-role constraints and inter-role constraints. The inter-role constraints are stored in distributed coordinators. If different coordinators have overlapping coordinatees, i.e., roles, 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 greatly reduce the complexity of an ARC system.

3.3 Reflective Russian Dolls (RRD)

Reflective Russian Dolls (RRD) [17] is a model of distributed object reflection based on rewriting logic. The model combines logical reflection with a structuring of distributed objects as nested configurations of meta-objects (a la Russian Dolls) that can reason about and control their sub-objects. In this formalism, a coordinator is an object with a distinguished attribute that holds a nested configuration of objects and messages. The nested configuration itself could consist of base-level objects or coordinators each with their configuration of coordinated objects. The rewrite rules that specify the behavior of a coordinator object control delivery of messages in its contained configuration as well as specifying how peer to peer messages are processed. Messages are taken from output of a sender object, or input to a receptionist from an external object, or possibly created. These messages can be immediately delivered to the designated receiver (placed in the input of a local object or put in the outgoing mail for an external object), delivered to another receiver, modified, reordered, replicated, or dropped.

RRD provides a very general coordination mechanism. In [21] a special form of RRD called policy based coordination (PBRD) was introduced. Here each coordinator has two additional required attributes: a policy attribute, and a policy state attribute, that maintains processing state. In this case the rewrite rules interpret the policy attribute, selecting a message to process and specifying what to do with it. Simple policies include ordering of message delivery, serializing requests, and recording a history of events. Policy languages can be simple tables, automata, or expressive functional languages. An example of PBRD coordination is the Policy And GOal based Distributed Architecture (PAGODA) for specifying systems of autonomous agents [24,22].

3.4 Feature Analysis

Computation model. Reo is a channel based language. Channels may be either synchronous or asynchronous. A channel is called synchronous if the pairs of operations on its two ends can only succeed atomically; otherwise it is called asynchronous. There is no shared global memory. Both ARC and RRD are based on the actor model of computation [1,16,2] with the coordinated objects being actors and the coordinators being meta-actors. Actors encapsulate their state and thread of control and communicate by asynchronous message passing. Meta-actors control the communication semantics of their base level actors.

Control. Coordination is imposed by a Reo circuit on connected components by determining when data can be accepted on input ports and when it can be taken from output ports, blocking components attempting write or read until the operation is available. The decision to connect to a port is made by the coordinatee, but once connected the coordinatee has no control over how the data is routed.

In the ARC model, role meta-actors intercept and control the delivery of base level messages. Formally, each base level action generates events that must be handled by the appropriate role before further base-level computation can take place. Role and coordinator meta-actors also communicate by events. The base-level actors have no active role in the coordination. However, roles are aware of the higher level coordinator and participate actively in their coordination. A novel aspect is that individual actors in a role are transparent to the coordinator layer. In the RRD model, coordination is exogenous at all levels. At each level, lower-level objects execute as if there were no coordination layer.

The actor model has a built in notion of communication / message delivery. This is modified by coordinators in ARC and RRD using reflective mechanisms. In contrast, Reo components have individual behavior but there is no built in communication semantics for collections of components. This is provided by Reo connectors.

Semantics. Reo has an operational semantics given by constraint automata (CA) [9] and a denotational semantics based on Timed Data Streams (TDS) [8,6]. In CA, states represent Reo configurations and transitions encode maximally-parallel stepwise evolution. Transition labels show maximal sets of active nodes and sets of data constraints. Timed data streams model the possible flows of data on connector ports, assigning a time to each interaction (input or output of a data element). The semantics of ARC coordinators, roles and actors is given by the composition of a state transition system that allows concurrent transitions and a concurrent constraint system that restricts the order and location of certain transitions. The operational semantics of RRD coordinators and components is a rewriting logic system, a state transition system that allows concurrent transitions. The denotational semantics is a set of interaction paths—sequences of interactions, both peer-peer and object-metaobject. It is derived from the event partial order generated by executions of the rewriting semantics. The relationship between timed data stream and interaction path semantics is discussed in Section 5.

Modularity and Compositionality. In Reo, more complicated connectors are made out of simpler ones. Nodes can be hidden by putting a box around a Reo connector, giving the connector a well-defined interface and making it a reusable entity. Both the CA and the TDS semantics are compositional—the behavior of a system can be constructed from the behavior of its constituents. The behavior of components as well as connectors can be

given using CA or TDS, and so, we may have the behavior of the whole system as a CA or a TDS.

The key structuring mechanism of ARC is the notion of role, with overall coordination layered on top of the per role coordination. ARC semantics is compositional when certain restrictions are obeyed by the configuration of roles and coordinators, i.e., neither roles nor coordinators share coordinatees.

The essence of RRD is the nested hierarchical structure of coordinators. This structure is preserved by basic composition operations. Event based semantics and interaction semantics are compositional both for pure actor systems and reflective systems—the semantics of a composition of objects and coordinators can be computed from the semantics of the parts (see [23,12]).

Specification. A Reo circuit may be specified by a constraint automaton. Then this constraint automaton can be compared with the constraint automaton obtained as operational semantics of a Reo circuit to check (bi)simulation or language equivalence. Temporal logics for specifying properties of Reo circuits are presented in [6], [10], and [15], with main focus on real-time, reconfiguration, and model checking, respectively.

In ARC there are two types of coordination constraints, namely intra- and inter-role constraints. For intra-role constraints, we use guarded action to specify when a message should be re-routed to another destination within the group, or re-ordering within an actor in order to satisfy the coordination constraints. In contrast, inter-role constraints are a set of boolean properties that the roles being coordinated must satisfy. Requirements for PBRD coordinators have been specified by informal constraints on the resulting interactions of the coordinated actors (see [21,22]). Behaviors of specific ARC and PBRD coordinators can be specified in rewriting logic. No formal logic has been developed or adapted to date for either ARC or PBRD.

Analyzability. Compositionality means that coordinators and coordinatees can be analyzed separately in any of the models. For Reo, regular model checking approaches can be adapted for constraint automata [15,10]. ARC's analyzability lies in the satisfiability and schedulability of composed inter-role and intra-role constraints. Although the satisfiability and schedulability in general are undecidable, certain techniques, such as graph theory, can be applied to identify infeasible situations. Furthermore, if the roles and coordinators are well partitioned, the complexity of constraint analysis can be reduced. The Maude rewriting logic language provides search and model-checking functions that can be used to analyze RRD systems. Use of policies expressed in restricted form can make coordinators easier to analyze.

4 Car Factory Case Study

In this section we look at how each of the three models addresses a particular coordination problem, namely coordinating different jobs in a factory. This example is taken from [20].

4.1 Specification

There are three factory jobs (called roles in [20]) to be coordinated: an assembler and some number of wheel and chassis producers. The requirements for job components (role players) are the following.

- An assembler receives car requests from a buyer, and parts (wheel or chassis) from producers. For each car request, it sends four part requests to wheel producers and one to a chassis producer. When four wheels and a chassis have been received it sends a car reply to a buyer.
- A wheel producer receives wheel requests and sends wheel replies.
- A chassis producer receives chassis requests and sends chassis replies.

The car factory system has one assembler, a, one chassis producer, c, and n wheel producers, $w_1, \ldots, w_n$. The assembler knows the chassis producer and one or more wheel producers, each producer (wheel or chassis) knows the assembler [5] . The assembler is the only receptionist (the only actor that can receive messages sent from outside the system). The requirements for the factory coordinator are

1. The ratio of chassis to wheel deliveries to the assembler is $1 : 4$.
2. The $1 + 4$ parts are delivered atomically.
3. Work is uniformly distributed amongst the wheel producers.

In the following subsections factory coordinators are described in Reo, ARC and RRD.

4.2 Reo Factory

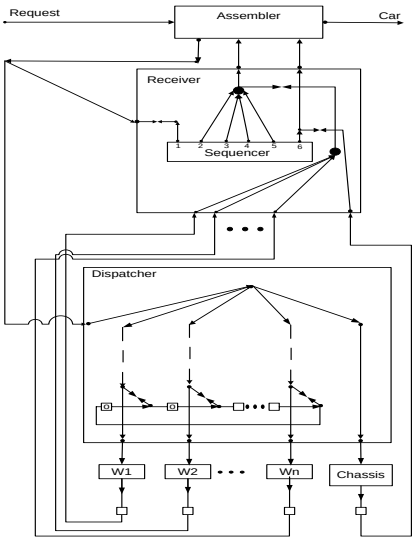


Fig. 2. Factory example using Reo

The actors—assembler, wheel producers, and chassis producers—are modeled as components. By putting an unbounded FIFO where an actor is connected to a Reo circuit, the inherent non-blocking and asynchronous behavior of actors is kept unchanged (i.e.,

[5] In the actor setting one actor must 'know' another in order to send a message. In a channel based setting, 'knows' means sending on a suitable port.

Reo connectors cannot block the actors when actors are sending messages). A Reo circuit to coordinate these actors that satisfies the three requirements is shown in Figure 2. Using boxes, we may distinguish two modules: *request dispatcher* and *part receiver* in the Reo circuit, which we call as Dispatcher and Receiver, respectively. The Dispatcher sends chassis requests to the chassis producer and incorporates a round-robin policy in sending requests to "four out of n" wheel producers (to satisfy Requirement 3). The produced parts (messages) go from the producer actors to the Receiver. A Sequencer is used in the Receiver to send the parts atomically to the Assembler, satisfying Requirements 1 and 2. The Reo circuit has been mapped to constraint automata where it was shown that the requirements are satisfied (the constraint automaton is not included in this paper for the lack of space).

4.3 ARC Factory

$\gamma_A(busy = false)$:

$P_1 : [\epsilon_{a.receive(carReq)}]$

$if\,(busy == true)$ `reroute` $(carReq, a, \alpha_{\perp_A}(t))$ $else$ $(busy = true)$;

$P_2 : [\epsilon_{a.send(buyer,car)}]$ $busy = false$;

$\gamma_W(x = 0)$:

$P_1 : [\epsilon_{w.send(a,wheel)}]$

$if(w \in \gamma_W \wedge w \neq \alpha_{\perp_W}(t))$ `become` $(\gamma_W(x++))$;

`tell` $(X = x) \rightarrow w.out\ (a, wheel)$ $\square$

$ask\ (X \neq x) \rightarrow$ `reroute` $(wheel, a, \alpha_{\perp_W}(t))$;

$P_2 : [\epsilon_{A.send(buyer,car)}]$ `become` $(\gamma_W(x = 0))$;

$P_3 : [\epsilon_{w_i.receive(wheelReq)}]$

$if\ (\exists j, 1 \leq j \leq n, s.t., |\mu_{w_j}| = \min_{1 \leq k \leq n} |\mu_{w_k}|)$ `reroute` $(wheelReq, w_i, w_j)$;

$\theta(X : Y = 4 : 1)$:

$[\epsilon_{\gamma_W.become\ (\gamma_W(x=0))} \cup \epsilon_{\gamma_C.become\ (\gamma_C(y=0))}]$ `become` $(\theta(X : Y = 4 : 1))$

Fig. 3. Factory example using ARC

The ARC specification of the car factory coordination is shown in Figure 3. γ_A, γ_W, and θ denote the assembler role, wheel role, and the coordinator, respectively. [6] Expressions of the form $[\epsilon_{action}]$ denote events that trigger role and coordinator's coordination actions, A $\square$ B denotes that either A or B will take place; and $|\ \mu_\alpha\ |$ represents the size of actor α's mail box. The intra-role coordination for the assembler role is to ensure that if its member actor is busy (represented by the role's state variable *busy*), the role will buffer further incoming requests by rerouting them to its sink actor, $\alpha_{\perp_A}$. If there is a deadline t associated with the request, the message will be tagged with t. Upon observing the assembler actor finishing a car, the role resets its busy state to `false`. The wheel role has a state

[6] As the chassis role has similar behavior to the wheel role, we omit its discussion.

variable x, initially 0, that tracks the number of wheels produced since the last delivery to the assembler. The wheel role not only synchronizes with the chassis role through the coordinator by the primitive `tell` and `ask` operations to ensure a 4:1 ratio, but also reroutes wheel requests to ensure that they are evenly distributed. The coordinator specifies the inter-role coordination requirement. In this example, it ensures that wheel role and chassis role' productivity must be a 4:1 ratio.

4.4 RRD Factory

A PBRD factory coordinator has the form

```
[FC : Factory | {_},
 policyState: (pending,wQ), policy: FactoryPolicy,
| in: iQ, out: oQ, in-a: aQ, out-x: xQ,
  up-a: auQ, dn-a: adQ, up-c: cuQ, dn-c: cdQ,
  up-w1: w1uQ, dn-w1: w1dQ, ..,up-wk: wkuQ, dn-wk: wkdQ]
```

The coordinatees have been replaced by message queue attributes representing their interfaces: `up-a` corresponds to the output queue of the assembler, `a`, `dn-a` corresponds to its input queue. Similarly for other `up/dn` attributes. In addition the coordinator exposes interfaces `in-a` for input to the assembler, and `out-x` for replies to external actors. A PBRD policy state consists of a queue of messages,`pending`, and a wheel actor queue, `wQ`. There are rules that unconditionally read messages from the input `in-a` and `up` interface queues and place them in `pending` (tagged with the interface name). There are five rewrite rules for the policy, `FactoryPolicy`.

r1. if `pending` has at least 4 wheel replies and at least 1 chassis reply addressed to the assembler a, remove 4 wheel replies and 1 chassis reply from `pending` and deliver them to a (put them in `dn-a`)

r2. if `pending` has a car request, deliver it to a

r3. if `pending` has a wheel request for some w, deliver it to the next wheel in `wQ` and rotate `wQ`

r4. if `pending` has a chassis request, deliver it to c

r5. if `pending` has a car reply put it in `out-x`

It is easy to see from the policy rules that the PBRD Factory coordinator satisfies the three requirements. In particular the only parts messages delivered to the assembler are by rule r1, and each delivery consists of 4 wheels and a chassis, thus guaranteeing the 4:1 ration (requirement 1) and atomicity (requirement 2). The only requests delivered to wheel actors are by rule r3, which uses a round robin policy, thus guaranteeing uniform load distribution (requirement 3) in the sense of the number of requests to any two wheel actors differ by at most 1 at any time. We have described a single level PBRD Factory coordinator. It is also possible to structure the coordination using a level of Role coordinators and an overall coordinator, emulating the ARC approach.

4.4.1 Discussion.

Although the three models use different basic coordination primitives, there is a clear correspondence in the organization. Requirement 1-2 are addressed by the Reo Sequencer

module, by the ARC coordinator rule plus the wheel rule P1, and by the PBRD rule r1. Requirement 3 is addressed by the Reo Dispatcher module, the ARC wheel role (P3) and the PBRD rule r3.

5 Semantic Foundations

In addition to comparing coordination models according to qualitative features, one can consider when coordinators represented in the different models are equivalent. For this purpose a common semantic foundation is needed. For the present, we focus on coordinating actor-like communication, that is asynchronous message passing. We assume an unbounded FIFO buffer at each connection point between a component and a Reo connector. We also assume Reo components send messages—pairs consisting of a target name and a data element [7]. Under these conditions we establish mappings between the TDS semantics of Reo components and connectors [8,6] and the Interaction Semantics of actors and meta actors [23,12].

5.1 Basic Definitions

We first define the two semantic domains and some auxiliary notation and give a small example.

Sequences. Following the Reo convention, we assume sequences are infinite and can thus be treated as functions from the natural numbers to the domain of sequence elements. We write $s(i)$ for the ith element of sequence s.

Timed Data Streams (TDS). A TDS over a 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 the non-negative reals. The semantics of a Reo connector with m ports is a set of m tuples of timed data streams, one for each port (i.e. an m-ary relation) [8].

Interaction Paths (IP). Given a set of object identities O, a data domain D, and a set of interfaces $IF = \{\phi_1, \ldots \phi_m\}$, an interaction is a triple (ϕ, o, d) where (o, d) is a message, with target o and contents d. An interaction path is a sequence of interactions. The semantics of an RRD coordinator is a set of interaction paths corresponding to its possible sequences of interactions.

The projection, $\pi(\theta, \phi)$, of an interaction path, θ, onto an interface ϕ is the subsequence of elements of θ with interface ϕ (preserving order). The function $ix(\theta, \phi)(j)$ returns the index of the jth element of $\pi(\theta, \phi)$ in θ. Thus if $ix(\theta, \phi)(j) = n$, then $\theta(n) = (\phi, o, d)$ for some (o, d), and there are j occurrences of interactions with interface ϕ in θ before n (since sequence indices start at 0). Given a correspondence ϕ_i to port i and letting $E = O \times D$, the projection $\pi(\theta, \phi_i)$ corresponds to the data stream on port i, and the function $ix(\theta, \phi_i)$ corresponds to the relative temporal ordering of events on port i.

Example. Consider the simple example of a coordination requirement to ensure that input to a receiver component alternates between data sent from two senders. A Reo connector meeting this requirement would have three ports, two for input from the senders and one for output to the receiver. The alternator semantics for the Reo connector is a relation Alt_{TDS}

[7] Although communication of Reo components is "untargeted", nothing prevents a connector from using information in the data to direct it. Dually, although actor messages are targeted, a coordinator in ARC or RRD may redirect it.

[8] These tuples are usually represented as a pair of tuples, one for input and one for output ports.

on TDS triples where

$$((a_1, \alpha_1), (a_2, \alpha_2), (a_3, \alpha_3)) \in Alt_{TDS}$$

just if for $i \in \mathbf{Nat}$

$$a_3(2i) = a_1(i), \quad a_3(2i+1) = a_2(i), \quad \alpha_1(i) < \alpha_3(2i), \text{ and } \alpha_2(i) < \alpha_3(2i+1).$$

A PBRD coordinator meeting this requirement has interfaces ϕ_1, ϕ_2 for messages sent by two sender objects, say o_1, o_2, and an interface ϕ_3 for messages to be delivered to a receiver object, o_3. The interaction semantics for a PBRD alternator are the interaction paths that satisfy Alt_{io} where

$$\begin{aligned}\theta \in Alt_{io} \Leftrightarrow\ & \pi(\theta, \phi_3)(2i) = \pi(\theta, \phi_1)(i) \wedge \pi(\theta, \phi_3)(2i+1) = \pi(\theta, \phi_2)(i) \wedge \\ & ix(\theta, \phi_1)(i) < ix(\theta, \phi_3)(2i) \wedge ix(\theta, \phi_2)(i) < ix(\theta, \phi_3)(2i+1)\end{aligned}$$

Thus, if we identify $\pi(\theta, \phi_i)$ with a_i and $ix(\theta, \phi_i)$ with α_i we see that the two relations correspond.

5.2 Factory Specification

Having introduced the semantic model, we can make the Factory Coordinator requirements more mathematically precise as constraints on the interaction paths θ of the coordinator semantics. We let $m, i, j, i', j', j_1, \ldots$ range over the natural numbers. The interfaces are (a, in), (a, out) (assembler communication with customers), (a, up), (a, dn) (assembler output/input), (c, up), (c, dn) (chassis output/input), and $(w_i, up), (w_i, dn)$ (ith wheel output/input), for $1 \leq i \leq n$.

Requirements 1, 2. Given that interaction paths are infinite, the notion of ratio of deliveries is not so simple to define. Thus requirements 1 and 2 are reformulated as: if any part is delivered to the assembler, the remaining parts of the $1 + 4$ set are delivered in a sequence that is not interleaved with any other deliveries. Namely, there is a function g mapping numbers to sequences of numbers such that if $\theta(m) = ((a, dn), p)$ where p is chassis or wheel (a part delivered to the assembler) then $m \in g(m) = [j_1, j_2, j_3, j_4, j_5]$ where $j_1 < j_2 < j_3 < j_4 < j_5$ and $\{d \mid (\exists 1 \leq k \leq 5)\theta(j_k) = ((a, dn), d)\}$ consists of one $chassis$ and four $wheels$. If $\theta(m') = ((a, dn), p')$ then either $g(m) = g(m')$ or $g(m) \cap g(m') = \emptyset$. For other m, $g(m)$ is the empty sequence.

Requirement 3. Uniform distribution of requests to wheel producers can be interpreted in at least two ways, one is essentially round-robin scheduling, the other is balancing the pending requests for each producer. These differ if the wheel producers have different production rates. The following formalizes the round-robin interpretation. If $\theta(m) = ((w_i, dn), wheel)$ (a wheel request delivered to wheel producer w_i), and m is the index in θ of the jth wheel delivery, then $i = j \ mod \ n$.

5.3 Mappings between TDS and IP

We define functions $tds2ip$ mapping timed data streams to sets of interaction paths, and $ip2tds$ mapping interaction paths to sets of timed data streams. The mapping of data sequences is one-to-one. The fact that the images of these mappings are sets is due to the

fact that for each stream or path there are a number of streams/paths that are equivalent in the sense that they represent different temporal views of the same underlying execution. We characterize the temporal views by ordering constraints and show that related streams satisfy the same ordering constraints.

We let D, O, IF be a data domain, set of object identifiers, and a set of m interfaces as above. We let $\tau = ((a_i, \alpha_i) \mid 1 \leq i \leq m)$ be a TDS tuple over $E = O \times D$ for a connector with m ports, and let θ be an interaction path over IF, O, D.

To define the mappings it is convenient to introduce the notion of stage in a TDS. The nth stage of data transmission of τ, $S(\tau)(n)$, is defined using auxiliaries $J(\tau)(n,i)$—the index of the remaining tail of α_i after the nth global time point and $N(\tau)(n)$—the set of ports active at the nth global time point as follows.

$$
\begin{aligned}
&J(\tau)(0,i) = 0\\
&N(\tau)(n) = \{i \mid \alpha_i(J(\tau)(n,i) \leq \alpha_l(J(\tau)(n,l)) \quad \text{for} \quad 1 \leq l \leq m\}\\
&J(\tau)(n+1,i) = J(\tau)(n,i) + \text{ if } i \in N(\tau)(n) \text{ then } 1 \text{ else } 0\\
&S(\tau)(n) = \{(i, J(\tau)(n,i)) \mid i \in N(\tau)(n)\}
\end{aligned}
$$

Thus $(i,j) \in S(\tau)(n)$ if data flows on the ith port at the nth global time point, $\alpha_i(J(\tau)(n,i))$. Note that if $(i,j) \in S(\tau)(n)$, $(i',j') \in S(\tau)(n)$, $n < n'$, and $(i'',j'') \in S(\tau)(n')$ then $\alpha_i(j) = \alpha_{i'}(j') < \alpha_{i''}(j'')$. Furthermore for any $1 \leq i \leq m$ and any j, there is some n such that $(i,j) \in S(\tau)(n)$, and if $(i',j') \in S(\tau)(n')$ with $n < n'$ then $\alpha_i(j) < \alpha_{i'}(j')$.

We restrict attention to semantic relations defining coordinator behavior to those specified by a (possibly infinite) set of timing constraints of the form $t(i,j) < t(i',j')$ and a set of constraints on the data streams. τ satisfies $t(i,j) < t(i',j')$ (written $\tau \models t(i,j) < t(i',j')$) just if $\alpha_i(j) < \alpha_{i'}(j')$ and θ satisfies $t(i,j) < t(i',j')$ (written $\theta \models t(i,j) < t(i',j')$) just if $ix(\theta, \phi_i)(j) < ix(\theta, \phi_{i'})(j')$. A set of constraints is satisfied if each element is satisfied. Here we do not further restrict the form of data constraints. Each TDS tuple, τ, or IP, θ, defines a set of temporal constraints, $C(\tau)$ or $C(\theta)$, characterizing its temporal view such that $\tau \models C(\tau)$ and $\theta \models C(\theta)$.

$$
\begin{aligned}
&C(\tau) = \{t(i,j) < t(i',j') \mid (\exists n < n')((i,j) \in S(\tau)(n) \wedge (i',j') \in S(\tau)(n'))\}\\
&C(\theta) = \{t(i,j) < t(i',j') \mid ix(\theta, \phi_i)(j) < ix(\theta, \phi_{i'})(j')\}.
\end{aligned}
$$

In a TDS tuple it is possible that more than one port is active at a given time, i.e. $S(\tau')(n)$ has more than one element for some n. Following [5], we interpret this as meaning that the two communications could have occurred in either order rather than requiring strict synchrony. We write $\tau' \sim \tau$ if τ' has the same ports and underlying data streams as τ, $S(\tau')(n)$ is a singleton for each n, and $\tau' \models C(\tau)$. Note that $\tau' \sim \tau$ implies that τ' satisfies any of the considered temporal and data constraints that τ does. By the non-zeno assumption for TDS, there are many such τ', each obtained by adding/subtracting small amounts to times at appropriate points in τ guided by the sets $S(\tau)(n)$.

To simplify the treatment of multiple 'simultaneous' communications we generalize interaction paths to sequences of multisets of interactions. A generalized interaction path stands for a (possibly infinite) set of interaction paths, each obtained by choosing some order for the elements of each multiset.

To define $tds2ip$ we first define $tds2ipg$ from timed data streams to a generalized interaction paths, then $tds2ip(\tau)$ is the set of interaction paths represented by $tds2ipg(\tau)$. $tds2ipg(\tau)(n)$ is the set of interactions that occur at the nth time point from the set of time streams of τ.

$$tds2ipg(\tau)(n) = \{(\phi_i, a_i(j)) \mid (i,j) \in S(\tau)(n)\}$$

$ip2tds(\theta)$ is the set of tuples of TDS such that the data part of the jth tuple component is the projection of θ onto the jth interface, and the time part is a monotonically increasing time sequence such that the ordering between interactions of θ is preserved.

$$ip2tds(\theta) = \{((\pi(\theta,\phi_i),\alpha_i) \mid 1 \le i \le m) \\ \mid (\forall 1 \le i,i' \le m)(\forall j,j')(ix(\theta,\phi_i),j) < ix(\theta,\phi_{i'},j') \Rightarrow \alpha_i(j) < \alpha_{i'}(j'))$$

Lemma. The mappings between TDS and IP satisfy the following.

$$(1)\ \theta \in tds2ip(\tau) \Rightarrow \theta \models C(\tau) \wedge \tau \in ip2tds(\theta) \Rightarrow \tau \models C(\theta)$$
$$(2)\ \theta \in tds2ip(\tau) \Rightarrow (\exists \tau' \in ip2tds(\theta))(\tau' \sim \tau)$$
$$(3)\ \tau \in ip2tds(\theta) \Rightarrow \{\theta\} = tds2ip(\tau)$$

Thus we see that we can move between the two forms of semantics preserving essential information.

Proof Sketch. For (1), assume $\theta \in tds2ip(\tau)$ and $(t(i,j) < t(i',j') \in C(\tau))$ then by the definition of $C(\tau)$ let $n < n'$ such that $(i,j) \in S(\tau)(n) \wedge (i',j') \in S(\tau)(n')$. If $\theta^* = tds2ipg(\tau)$, then $ix(\theta^*,\phi_i)(j) = n$, $ix(\theta^*,\phi_{i'})(j') = n'$ and $\theta^* \models (t(i,j) < t(i',j')$ as does any flattening of θ^*. Now assume $\tau \in ip2tds(\theta)$ and $(t(i,j) < t(i',j') \in C(\theta))$. Then $ix(\theta,\phi_i)(j) < ix(\theta,\phi_{i'})(j')$ and by definition of $ip2tds$, $\alpha_i(j) < \alpha_{i'}(j')$. For (2), the linearizing map used to obtain θ from $tds2ipg(\tau)$ can be used to transform τ to a linear form $\tau' \sim \tau$ satisfying the mapping conditions. For (3), note that $S(\tau)(n)$ is a singleton for any n.

6 Conclusions and Future Work

Each of the models is clearly highly expressive. The Reo model is more mature, with several formal semantics and tools for analysis. Reo is closer to being a programming model, while RRD focuses on more abstract specifications. The ARC model is aimed at coordination of resource usage and QoS goals while RRD has focused on logical communication constraints, as has much of the Reo work. All three models provide for user definable coordination behavior, but in different ways: channel behavior (Reo), coordinator events (ARC), coordinator rewrite rules (RRD). Although channels and messages seem very different operationally, denotationally they have similar semantics.

There are a number of topics for future work. Preliminary work indicates that Reo specifications as CA can be used as a policy language for PBRD and that ARC can be embedded fairly naturally into RRD. These mappings need to be worked out in detail. The full generality of rewriting logic and RRD make it difficult to give simple mappings from RRD to Reo or ARC. Logics for specification and reasoning about coordination are of great interest. Do the logics developed for Reo work more generally? Are new logics needed to express end-to-end properties emerging from coordination? An important topic is

developing methods to combine coordination rules for different concerns: communication constraints, timing, resource usage, etc., and to assure safe composition.

References

[1] Agha, G., "Actors: A Model of Concurrent Computation in Distributed Systems," MIT Press, 1986.

[2] 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.

[3] Arbab, F., *What do you mean, coordination?*, in: *Bulletin of the Dutch Association for Theoretical Computer Science, NVTI* , 1998, pp. 11 – 22.

[4] Arbab, F., *Reo: A channel-based coordination model for component composition*, Mathematical Structures in Computer Science **14** (2004), pp. 329–366.

[5] Arbab, F., *A behavioral model for composition of software components*, L'Objet **12** (2006), pp. 33–76.

[6] Arbab, F., C. Baier, F. de Boer and J. Rutten, *Models and temporal logics for timed component connectors*, in: *IEEE International Conference on Software Engineering and Formal Methods* (2004), pp. 198–207.

[7] Arbab, F., C. Baier, J. J. Rutten and M. Sirjani, *Modeling component connectors in Reo by constraint automata (extended abstract)*, in: *FOCLASA'03*, ENTCS **97**, 2003, pp. 25–46.

[8] Arbab, F. and J. Rutten, *A coinductive calculus of component connectors*, in: *WADT'02*, LNCS **2755**, 2002, pp. 34–55.

[9] Baier, C., M. Sirjani, F. Arbab and J. Rutten, *Modeling component connectors in reo by constraint automata*, Science of Computer Programming **61** (2006), pp. 75–113.

[10] Clarke, D., *Reasoning about connector reconfiguration ii: Basic reconfiguration logic*, in: *FSEN05*, Electronic Notes in Theoretical Computer Science, 2005.

[11] De Nicola, R., J. Katoen, D. Latella and M. Massink, *Towards a logic for performance and mobility*, in: *3rd Workshop on Quantitative Aspects of Programming Languages, QAPL05* (2005), pp. 132–146.

[12] Denker, G., J. Meseguer and C. L. Talcott, *Rewriting semantics of distributed meta objects and composable communication services*, in: *Third International Workshop on Rewriting Logic and Its Applications*, ENTCS **36** (2000).

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

[14] Gorrieri, R. and C. Hankin, *Theoretical aspects of coordination languages (foreword)*, in: *Theoretical aspects of coordination languages*, TCS **192**, 1998, pp. 163–165.

[15] Klüppelholz, S. and C. Baier, *Symbolic model checking for channel-based component connectors*, in: *FOCLASA'06*, 2006.

[16] Mason, I. A. and C. L. Talcott, *Actor languages their syntax, semantics, translation, and equivalence*, Theoretical Computer Science **220** (1999), pp. 409 – 467.

[17] Meseguer, J. and C. L. Talcott, *Semantic models for distributed object reflection*, in: *European Conference on Object-Oriented Programming, ECOOP'2002*, LNCS **2374**, 2002, pp. 1–36, invited paper.

[18] Nicola, R. D., G. Ferrari and R. Pugliese, *KLAIM: A kernel language for agents interaction and mobility*, IEEE Transactions on Software Engineering **24** (1998), pp. 315–330.

[19] Picco, G., A. Murphy and G.-C. Roman, *LIME: Linda meets mobility*, in: *21 Int. Conf. on Software Engineering*, 1999, pp. 368–377.

[20] 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: *Coordination Models and Languages*, LNCS **4038**, 2006, pp. 247–265.

[21] Talcott, C., *Coordination models based on a formal model of distributed object reflection*, in: *1st International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord 2005)*, 2005.

[22] Talcott, C., *Policy-based coordination in pagoda: A case study*, in: *2nd International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord 2005)*, 2006.

[23] Talcott, C. L., *Composable semantic models for actor theories*, Higher-Order and Symbolic Computation **11** (1998), pp. 281–343.

[24] Wirsing, M., G. Denker, C. Talcott, A. Poggio and L. Briesemeister, *A rewriting logic framework for soft constraints*, in: *Sixth International Workshop on Rewriting Logic and Its Applications*, ENTCS (2006).