International Journal of Distributed Sensor Networks, 3: 175–199, 2007

ISSN: 1550-1329 print/1550-1477 online
DOI: 10.1080/15501320701205068

Taylor & Francis
Taylor & Francis Group

A Coordination Model for Improving Software System Attack-Tolerance and Survivability in Open Hostile Environments

SHANGPING REN and YUE YU

Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA

KEVIN A. KWIAT

Information Directorate, Air Force Research Laboratory, Rome, NY, USA

JEFFREY TSAI

Department of Computer Science, University of Illinois at Chicago, Chicago, IL, USA

This paper presents a coordination model that contains three active entities: actors, roles, and coordinators. Actors abstract the system's functionalities while roles and coordinators statically encapsulate coordination constraints and dynamically propagate these constraints among themselves and onto the actors. A software system's attack-tolerance and survivability in open hostile environments are enhanced through appropriate constraint propagations and constraint enforcements.
The role represents a group of actors that share the same set of behaviors declared by the role. Coordination and coordination constraints in the model are categorized into two classes: inter-role coordination and intra-role coordination. The coordinators are responsible for inter-role coordination; while the roles are not only abstractions for a set of behaviors they also coordinate the actors which share the same role. This setting implies that both the coordination constraints and coordination activities are decentralized and distributed among the coordinators and the roles. The decentralization not only shields the system from single point of failures, but also provides a foundation that survivable feedback loops can be built upon. The survivable feedback loops presented in the model resist the contamination of the system by faulty elements and thereby protect the whole system from being broken down by single failures.

Keywords Sensor Networks; Coordination Model; Attack-Tolerance; Distributed Voting; Probabilistic Analysis

1. Introduction

Fault tolerance creates a stringent boundary within the information system beyond which the effects of faults shall not propagate; yet cyber defenses are layered. Cyber defenders realize that some defenses will be penetrated due to (then unknown) vulnerabilities;

Address correspondence to Shangping Ren, Illinois Institute of Technology, Department of Computer Science, 10 W. 31st Street, Chicago, IL 60616, USA. E-mail: ren@iit.edu

however, other defensive layers should not have the same vulnerability and thus will withstand the attack and remain intact. Once a defensive layer is breached, the attacker-induced faults can then propagate but the other layers of defense are intended to insulate the user from the effects of the attack. "Defense in depth" is a mainstay of protecting networked information systems from attack and fault tolerance techniques are applied to a defensive layer for the containment of faults.

According to Degani's book on computer interfaces [4, 25], we all have an internal model of how things ought to work and a fixed and conventional set of expectations telling us what our interaction with a machine "should" result in. The results are usually beneficial; yet, while anticipating the benefits of computers we simultaneously cope with the anxiety of the inevitable computer glitches: personal computers hang, on-line transaction systems become inexplicably non-responsive, and frequently visited web pages are suddenly unavailable. Our notion of how computers work for us is therefore tainted by the opposing notion of how frequently they fail. In the context of ubiquitous computing, the sheer number of computers involved can only mean that likelihood of failure cannot be remote. Blending conflicting notions of positive and negative computer performance is also pointed out by Degani when he says:

> *These machines fascinate us because they are powerful, dutiful, precise, unemotional, and extend our abilities beyond our human reach. Yet at the same time, we have a deep fear of being betrayed and hunted by the very machines that we create and nurture*.

Of course, in critical computer applications, such as command and control, the fear of machine betrayal is drastically reduced because they are built according to principles for achieving high system dependability. Dependability is that property of a computing system that allows reliance to be justifiably placed on the service it delivers [14], and the following equation captures dependability's quantification:

$$Dependability_{system} = Pr\{NoFault\} + Pr\{CorrectOperation \mid Fault\} \times Pr\{Fault\}$$

where $Pr\{X\}$ denotes the probability of event X occurrence.

The above equation indicates that increasing the dependability of a system means increasing the terms on the right hand side. Attempting to make $Pr\{NoFault\} = 1$ is possible when the system is composed of the best, most reliable, and therefore expensive components and that the system undergo extensive design reviews and testing to remove any faults prior to fielding the system. Achieving $Pr\{NoFault\} = 1$ is unrealistic; no one can reasonably expect a component of even moderate complexity to not have some potential to fail. This is where the second term compensates for the first: maintaining correct operation given the condition that a fault has occurred defines fault tolerance.

It is important to note that the above discussion is for faults that we can predict. Faults induced by attackers [10], however, cannot be treated in this way because time is on the side of the attackers: they probe a system for weaknesses and decide when to create a fault that will lead to failure(s). Nomenclature from fault-tolerant computing says that a fault leads to failure(s), and the effects of a failure are exhibited as an error. In cyber defense, attackers probe the system for vulnerabilities. Once such vulnerability is exposed to the attacker and we say that the attacker has induced a fault. Therefore, our approach to faults is not to predict them but to prevail over them. By distributing the security and fault-tolerance

policies among mutually checking components, we create survivable feedback loops to support a form of self-adaptivity that transforms fault tolerance to attack tolerance.

The remainder of the paper is organized as follows: Section 2 presents the ARC (Actor, Role, and Coordinator) model in which the security and fault-tolerance policies are distributed among mutually checking components (i.e., roles and coordinators). The roles and coordinators, on one hand, are separated from the components (i.e., actors) that perform the system's regular functionalities. On the other hand, they enforce and monitor the specified policies on the actors. Section 3 presents multiple survivable feedback loops built in the ARC model and discusses how the ARC model may self-heal from attacks on individual actors, roles, and coordinators. Section 4 presents an application of the ARC model in multi-dimensional sensing where coordination between data availability and consistency constraints is needed. The related work is presented in Section 5. At the end, we conclude our work and point out future research directions in Section 6.

2. The ARC Model

Networked computers form the information systems on which people now rely, both in critical national infrastructures and in private enterprises. Today, many of these systems are far too vulnerable to cyber attacks that can inhibit their functioning, corrupt important data, or expose private information. "Attack tolerance" strives to maintain system liveliness and safety and has become a key research issue in the cyber defense research community.

According to Laprie's dependability definition [12], a system *failure* occurs when the delivered service deviates from fulfilling the system function; while an *error* is that part of the system state which is *liable to lead to subsequent failure*. The *fault* is the cause of an error.

If a system is attacked successfully, then the system will eventually be in a faulty state. The faulty behavior (i.e., a failure) may not manifest immediately. However, for an attack to be successful, we say that such an attack was preceded by a fault. This indicates that each successful attack (A) implies one or multiple fault(s) (F) i.e., $A \rightarrow F$. Logically, if we can prevent all the faults, we can survive all the attacks, i.e., the contra-positive of the implication, $\neg F \rightarrow \neg A$, should hold. Unfortunately, real computer systems in general are too complex to undergo enough scrutiny and testing to assure that all faults can be prevented. Therefore, we assume instead that some faults will be created from a successful attack. However, with the proper use of fault tolerance, the effects of the faults (i.e., errors) will be absorbed, not propagate beyond a known boundary, and the system will react accordingly (i.e., $\neg F$). It, hence, follows that $\neg F \rightarrow \neg A$ and the system we designed is self-adaptable and survivable to attacks.

Unlike traditional security measures with which the central control and administration are sufficient, survivability in open hostile environments must address highly distributed, dynamic, and unbounded environments that lack central control and unified policies [22, 19, 21]. To overcome this challenge and ensure software system dependability in open hostile environments, a good model that captures the characteristics of the system and the environment becomes an essential factor.

As pointed out in Dr. Randell's "Facing up to Faults" Turing Memorial Lecture [17], ". . . multiple activities in a decentralized system will often not simply be competing against each other for access to some shared internal resource, but rather will on occasion at least be attempting to cooperate with each other, in small or large groups, in pursuit of some common goal." As critical information systems emerge from a "closed castle" into distributed paradigms, the cooperation among distributed elements (which compose the larger cyber systems) inevitably become the focus of such systems. The ARC (Actor, Role, Coordinator) coordination model was developed to model an open distributed

system's non-functional requirements (or QoS requirements in general), such as its survivability and attack-tolerance requirements [18].

More specifically, the ARC model has the following characteristics:

- The Actor model is used to abstract the concurrent computational part of a distributed cyber information system, while an independent coordination model is developed to address the individual composing entities' cooperation or coordination. Further, the general QoS requirements and particular, survivability, and attack-tolerance requirements are realized through specific coordination among the asynchronous entities.
- The concept of *role* is introduced into the coordination model. The role provides an abstraction for coordinating behaviors that may be shared by multiple actors and also provides for localized coordination among its players.
- Coordination in our model is divided into inter-role and intra-role coordination to ensure clearer separation of responsibilities and reduce the complexity of individual coordination entities. This division further ensures that both the coordination constraints and coordination activities are decentralized and distributed among the coordinators and the roles.
- The survivability and attack-tolerance requirements are mapped to coordination constraints that are transparently imposed on actors through message manipulations carried out by the roles and coordinators.

Conceptually, the ARC model is composed of three layers as illustrated in Fig. 1. The separation of concerns is made apparent through the relationships between the layers. The actor layer is dedicated to functional behavior and is oblivious to the coordination enacted in the role and coordinator layers. The roles and coordinators occupy similarly-named layers, but together these two layers are responsible for imposing the coordination and QoS constraints among the actors. This dual-layer is oblivious to the actor layer. The coordinator layer is dedicated to inter-role coordination. The role layer serves as the bridges between the actor layer and the coordinator layer. The role layer may therefore be viewed

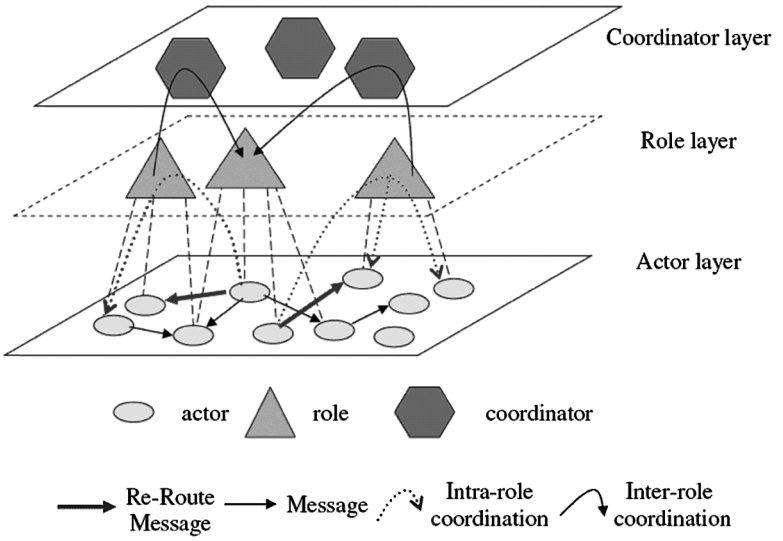


FIGURE 1 The ARC model.

from two perspectives: from the perspective of a coordinator, a role enables the coordination of a set of actors that share the static description of 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. The roles in the role layer and the coordinators in the coordinator layer are active state-based objects that enable the coordination policies within an application to adapt over time. While actors communicate via messages that are subject to delay, the information required by roles and coordinators is communicated via atomic events that are processed atomically by all interested roles and coordinators.

2.1 Actors

Actors in our ARC model are based on the actor model in [1]. More specifically, actors are active objects: they have states and behaviors. The states and the current behavior of the actors decide how they process messages (operations). Actors communicate with each other through asynchronous message transfer. Unprocessed messages are buffered at the receiving actors' mailboxes. An actor's state can only be changed by the actor itself, and state changes only occur during the processing of messages. The active thread within the actors will continuously process messages whenever their mailboxes are not empty (Fig. 2). Such an actor abstraction provides a unified representation of system resources because all functional processes, such as networks, network protocols, memories, services, and application processes (in-house-developed, or off-the-shelf products), are treated as actors. Furthermore, each actor can perform three different types of primitive operations: *send* messages, *create* (new actors), and *become* (to change its behavior). Corresponding to these operations are events that are observable by other entities in the systems.

2.2 Roles

Roles serve two purposes. First, roles provide static abstractions (declarative properties) for functional behaviors that must be realized by actors. Coordination based on roles is therefore relatively stable, even though the underlying actors may be numerous and dynamic. In addition, roles actively coordinate actors who play roles towards satisfying

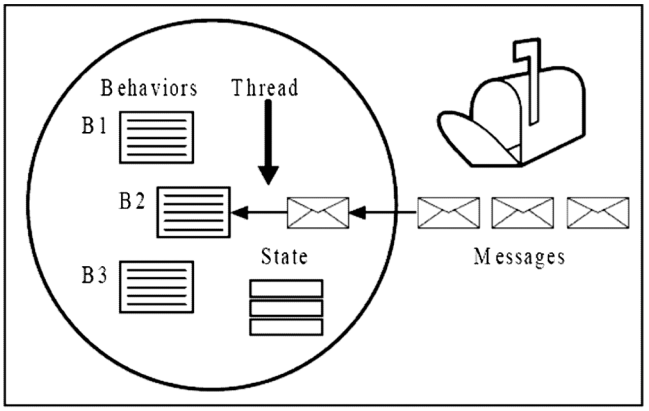


FIGURE 2 An actor.

fault-tolerance requirements. The intra-role coordination (that is played out by the actors) that is coerced by roles complements the inter-role coordination enacted by coordinators.

Every role must have a distinct purpose. This assumption disallows overlapping roles thereby eliminating the possibility that a role may be replaced by a set of other roles that might possibly create conflicting constraints upon an actor. These conflict-free constraints are based on the underlying actor model: each actor has only a single thread of control and hence at any given time, it can play only one role. More precisely, let $B(\gamma)$ denote the actor's functional behaviors declared by role γ, $B(\alpha)$ denote the functional behaviors provided by actor α, Γ and A denote the set of roles and actors in the system, respectively, and F is the actor to role assignment function. At any given time, well-defined roles and actors in a system must satisfy the following requirements:

- Roles are exclusive: the role declared behaviors do not overlap, i.e.,

$$\forall i \neq j, B(\gamma_i) \cap B(\gamma_j) = \varnothing \tag{1}$$

- Roles are exhaustive: all actors belong to one of the roles, i.e.,

$$\left(\bigcup_i^n B(\alpha_i) = \bigcup_j^m B(\gamma_j)\right) \wedge (\forall \alpha \in A, \exists \gamma \in \Gamma : B(\alpha) \subseteq B(\gamma)) \tag{2}$$

- Roles are repetitive: repeated actor behaviors replicate the assignment of the actor to the same role, i.e.,

$$\forall \alpha, \beta \in A : \quad B(\alpha) = B(\beta) \Rightarrow F(\alpha) = F(\beta) \tag{3}$$

- Each actor only plays one role at any given time, i.e.,

$$\forall i, j : B(\alpha) \cap B(\gamma_i) \neq \varnothing \wedge B(\alpha) \cap B(\gamma_j) \neq \varnothing \Rightarrow i = j \tag{4}$$

As active objects, roles have states, and based on its state coupled with a set of intra-role constraint policies, a role actively coordinates the actors sharing that role and maintains the integrity of itself. In addition, the roles are responsible for propagating constraints to actors and events to coordinators (the discussion of coordinator is in section 2.3). The intra-role constraint policies are role-state dependent. Hence, by changing role-states, we can have different and adaptive policies applied on actors. The role-state changes are triggered by observed events. Furthermore, the constraint policies are applied to actors through manipulating messages. Hence, the actions the roles may take can be: *re-route* a message to different actors within the role, *broadcast* a message to all group members with the role, or *block* a message temporarily from dispatching to the receiving actor. A template of role specification is given in Fig. 3.

The declarative criteria in the roles not only abstract the behaviors of actors, but also present a static interface to coordinators. Coordinators, therefore, do not have to directly coordinate actors, but implicit groups of actors, i.e., through roles. Although in an open environment actors are very dynamic, they join or leave the system frequently. With the abstraction created by roles, coordinators are refrained from such dynamics. The roles also contain a set of policies that govern over all the actors that play the role. For instance, if

```
Role role1(T1 p1) {
 state {
   T v; // state variable }

 criteria {
   property_A;
   // admissible criteria for actors
   // playing the role }

 policy {
   [eventB && property_X] {
     block(m1); // block m1
     v = newValue; // state update }}
}
```

FIGURE 3 A role template.

we use a replication mechanism to protect a data buffer against possible faults, we create multiple actors performing as data-buffers. In order for all these actors to be consistent, we have to meet synchronization constraints for any data written to or data read from the buffers. Therefore, we can define a *SyncBuffer* role which has a policy that all messages sent to any buffer actor that are of the *SyncBuffer* role must be broadcast to other *SyncBuffer* actors, and all messages within the *SyncBuffer* group, must be processed "atomically." Such constraints are transparent to actors. Figure 4 illustrates the *SyncBuffer Role*, where a message sent to a *SyncBuffer* actor is broadcast to all the role members. Figure 5 is the *SyncBuffer Role* specification.

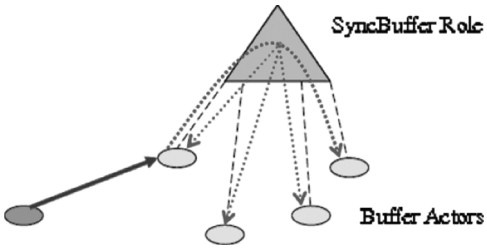


FIGURE 4 SyncBuffer.

```
Role SyncBuffer() {
 criteria {
  data == get(put(data));}

 policy {
  broadcast(put(data));
  broadcast(get());
  atomic(put(data));
  atomic(get());  }
}
```

FIGURE 5 SyncBuffer role specification.

For example, to eliminate a bad actor that tries to overflow the buffer by sending it highly frequent messages, a new policy that limits the put frequency can be added into the SyncBuffer role.

In the ARC model, coordination and constraint policies are distributed among coordinators and roles. One of the obvious advantages of such a distribution, other than avoiding the single point of failure and to control scalability, is localization of a fault's impact—if the role becomes faulty, then its only effect is upon those actors that play the role.

Another aspect of the roles is that the roles in the ARC model are active and have a set of supporting operations that not only maintain the role's integrity, membership integrity, and policy integrity, but also propagate constraints to actors, and events to coordinators.

As roles have their own policies, the supporting operations can check for compatibility between their own policies and with those the policies propagated from coordinators. This gives another level of protection against faulty coordinators.

2.3 Coordinators

Similar to the roles and actors, coordinators also have states and are active. They are able to observe events and make corresponding state adaptations. The declarative constraint policies are state-based and apply to roles only. The actors and coordinators are mutually transparent: though changes on actors or coordinators may impact each other, such impacts are only passed through roles.

Generally, in an open environment, the number of functional entities (i.e., actors), can be very large and quite dynamic. However, the functional categories of the actors in a system (i.e., the roles that actors play), are relatively small in number and tend to be stable. Hence, from a coordination perspective, role-based coordination is lightweight and more scalable compared with individual functional entity-based coordination especially in open systems. Furthermore, as the roles shield the dynamics of the actors, the logic of role-based coordination becomes simpler and more reusable.

It is worth pointing out that coordination in the ARC model is mostly concerned with synchronization among distributed entities and this echoes the main focus of most existing coordination models [7, 26]. For instance, in a battlefield, there are three roles, *AirForce, GroundForce*, and *BackupForce*. One scenario of coordination of these three roles is: *AirForce* must reach the destination at least one hour before the *GroundForce* or the *BackupForce* must be deployed as *AirForce*. The active, stateful coordinators provide the expressiveness to handle such complex coordination requirements.

A template of a coordinator specification is given in Fig. 6.

Informally, the constraint section defines a set of constraints. Each constraint can involve several roles. A constraint can be instantiated (i.e., "told") upon the role(s) when a coordinator handles an event in its policy section. Policies are guarded by a specification of the event and may be accompanied by a supporting boolean condition involving states and parameters. The guard for a policy is evaluated each time an event occurs, and although all applicable policies are processed in an arbitrary order, they maintain atomicity to the outside world. A policy may involve constraint updates and/or state updates. A major difference between the role and coordinator language definitions is that the roles have declarative criteria for their member actors and the policies within roles are enacted upon actors, while the coordinators have declarative constraints for their underlying roles and the constraint policies in the coordinators are imposed on the related roles.

```
Coordinator C[role1, role2](T1 p1) {
 state { T v; }

 constraints {
  constraint1 {
   role1(property_X1);
   role2(property_Y1) }

  constraint2 {
   role1(property_X2);
   role2(property_Y2) }
 }

 policy {
  [event1 && property1] {
   tell(constraint1);
   // propagate constraint1

   v := v + 1 // state update
  }
  [event2 && property2] {
   tell(constraint2);
   v := v + 2
  }
 }
}
```

FIGURE 6 A coordinator template.

2.4 Messages and Events

There are two types of communication media: messages and events. Messages are communications among actors and are processed by operations that the actors provide. Computations are carried out by processing messages. Events, on the other hand, are "happenings." Events can happen at actors, roles, or coordinators. However, as roles are transparent to actors and coordinators are transparent to roles, events at a higher level should not be of interest to the lower-level entities. The root cause of events can be traced to actors processing messages. As discussed in Section 2.1, the actors only have three primitive operations, i.e., *send*, *create*, and *become*, and these operations are the root causes of events.

In contrast to messages in the system, which are point-to-point in the actor model, events are broadcast. The roles and coordinators can filter out those events that are not of interest to them. By processing messages, actors may change their states and behaviors; similarly, by observing events, roles and coordinators may change their states, which in turn may cause their state-dependent policies to change accordingly. A policy change will then be propagated down to further influence other entities' behaviors.

3. Built-in Survivable Feedback Loops

From the discussions in previous sections, it is not difficult to see that the ARC model is a layered model—from coordination abstraction (coordinators), to functional abstraction and location coordination abstraction (roles), and to functional implementation (actors), with each layer having different responsibilities and communication schemes. The communication between layers is through events and propagation of constraints. Specifically, events are propagated up from actors to roles, and then further on to coordinators.

Constraints, on the other hand, are propagated from coordinators to roles and further down to actors. Figure 7 illustrates the dynamic propagation among the three types of entities in the ARC model.

As the figure depicts, there are two closed feedback loops. Loop 1 is between actors and roles. While processing messages, the actors may generate events; by observing events, roles may change their constraints and exclude "bad" (i.e., failed) actors from the group by re-routing messages out from the bad actors. In the ARC model, actors are functional units that implement the functionalities of the system. They are reactive such that only messages can trigger their activities. Hence, starving actors from messages effectively eliminates the bad actors from further impact to the system.

A similar loop exists between the roles and coordinators (Loop 2) in which the coordinators may exclude bad roles. In addition, because the roles also have their own policies they can, through compatibility checking, reject constraints from bad coordinators, and confine bad policies from propagating from to the actors beneath these coordinators. In particular, consider a scenario in which a local coordination policy within role *R* states, for instance, that each actor playing the role must finish processing a message of type *M* within *t* time units. Assume that actor *A* playing the role was attacked and eventually behaves differently from what the role *R* expects by taking longer than *t* time units when processing the message of type *M* (which could indicate the actor is exhibiting faulty behavior).

From *R*'s perspective, all the member actors have the same behaviors and are able to perform the same operations. By observing actor *A*'s faulty behavior, the role re-routes the messages targeted to *A* to other group members. By eliminating messages delivery to the infected actor *A*, actor *A* is starved and effectively excluded from the system. Other than from role *R's* perspective, such an elimination is transparent from the system.

The above scenario assumes that the role is not faulty and is able to observe unexpected behaviors. However, the role might itself become faulty. In this case, the role behaviors will deviate from what the coordinators expect and cause the coordinators to re-enact a role that satisfies the coordinators expectations. The semantics of the ARC model assures that each actor can play only one role at any given time, so the creation of a new role automatically removes the faulty role's control over the underlying member actors. However, as the intra-role policies are encapsulated and role-state-dependent, the newly-created role may not be implanted by the coordinator with those policies that were contained in the faulty role. Nevertheless, the new role can at least propagate the coordinator's constraints down to the actors. In other words, some local policies are removed from the system that may have effects on the system. These effects are localized and can only affect the member actors.

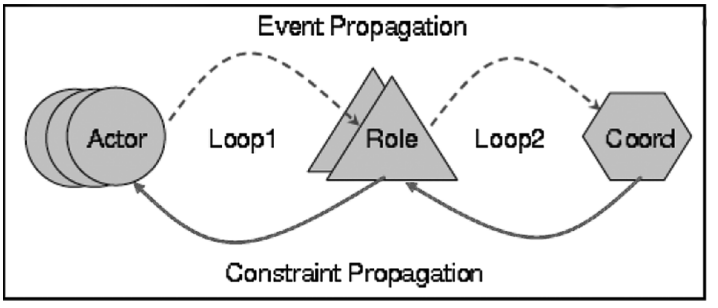


FIGURE 7 Survivable feedback loops.

When a coordinator becomes faulty, one scenario is that the faulty coordinator issues different or contradictory constraint policies. If all or majority of the coordinated roles (distributed voting scheme [11] is applied to find the majority) find the policies are incompatible with their local policies, it is highly likely that the coordinator is faulty. The roles can refuse further propagation of the faulty coordinator's rules and hence prevent the fault from spreading into other areas of the systems.

4. Coordination in Multi-dimensional Sensing—A Case Study

4.1 Background and Problem Description

To tolerate faults that might happen on individual sensors, such as radar devices, a widely used approach is to replicate the sensors. In certain situations, we may even deploy different types of sensors to ensure data integrity. For instance, to detect a foreign object, an infrared sensor and a wave sensor may be deployed in a protected region. Data sensed from different sensors must be correlated in time to make coherent identifications of the foreign object. In time critical systems, such as in a battlefield, data sensed from an environment has timeliness attributes associated with it, meaning that there is a time after which the data is of no use.

However, as argued by Dr. Lee in his invited talk [13], precise timing estimation of software execution time in an embedded networked system is impossible. However, what we may know are the statistics of time within a range. For instance, the moment an enemy plane emerges into a region, it usually takes a non-faulty radar within t_1 to t_2 seconds to detect it and transmit the information to a control center. In other words, normally, the command and control (C^2) center should receive the plane information within the $[t_1, t_2]$ time interval; however, *exactly* when may only be known statistically.

Consider a military setting in which two types of sensors are deployed in a region to monitor potential enemy aircrafts. One type is an infrared (IR) sensor used for producing thermo graphic images. Another type is a radio wave (RW) sensor used for measuring speed of unidentified flying objects. The IR sensors produce clear and reliable data. However, due to electromagnetic interferences and territorial circumstances, RW sensors produce less reliable data and hence it becomes necessary to get a consensus from other RW sensors deployed in the region. *Furthermore, in order for a soldier or* command and control (C^2) center *to take critical actions, the data from two different sources (IR and RW) must be coherent—not only they provide the correlated information (if IR gives an image of an airplane, the speed detected by RW must be within the jet flying speed limit), but also such information arrives at the requester within a limited time frame* Δ. The scenario is depicted in Fig. 8.

4.2 The General Design

Given the problem description, it is easy to see that the sensors could be organized into two disjointed roles, namely IR and RW. The constraints to ensure the semantic coherence and timing relation between the two different types of data obtained from IR and RW are part of the coordination among roles. The pseudo codes for the two roles are given in Fig. 9 and Fig. 10, respectively. In the following, we assume that a message between a sensor and the data requester – msg – is a tuple (tag,cv,aid), where the tag gives the timestamp for the message, cv is the communicative value of the message, and aid uniquely identify the receiver actor.

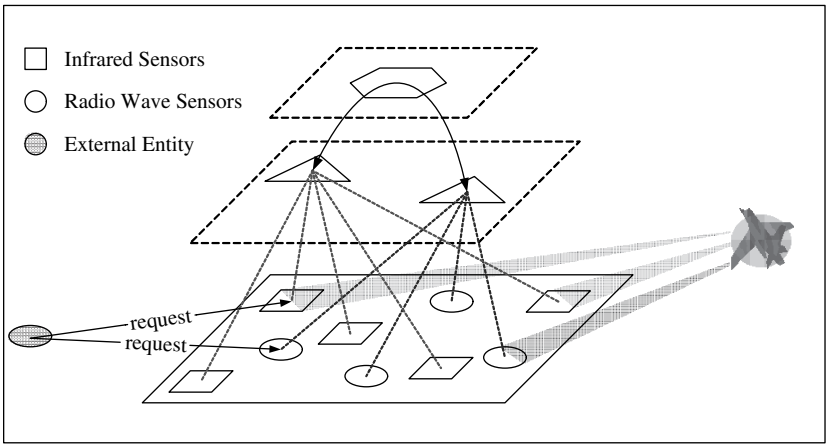


FIGURE 8 An entity is sending messages to two types of sensors requesting thermo graphic image and speed of an unidentified flying object.

```
Role IR() {
 state {
   AidList A;// a list of actor ids for multicast
 }
 criteria {
   (a is an IR sensor) ∧ a.C(t)= 1;//C(t) is the credibility function
 }

 policy {
  /*an event indicating that a request msg is sent to an actor under the role*/
  [request(msg)]{
     reroute(msg,A);//multicast msg to actors in A
  }
  /*an event indicating that a reply msg is sent from an actor under the role*/
  [reply(msg)]{
    if(tell(msg.tag)=false)
      block(msg);
  }
 }
}
```

FIGURE 9 IR role.

- IR: A criterion for an actor to be a member of this role is characterized by its declaration as an Infrared sensor and its credibility function $C(t)$ being a constant function equal to 1 ($C(t)$ is introduced in the next subsection, it basically states that the vote given out by the actor is always trustable). When an external actor sends a request to an actor under this role, the role multicasts the request to a list of actors under it. When an internal actor sends a reply, since any actors under this role is trustable, the role only needs to check if the tag of the message satisfies the timing constraints in the coordinators. Note that in Fig. 9, tell(msg.tag) sends the time of the message to all constraint stores to check for consistency.
- RW: Any actor who is a radio wave sensor and whose credibility function not constantly equal to 1 will be in this role. Since actors are untrustworthy (i.e., they may have failed), majority voting protocols will be used. When an external actor sends a request to an actor under this role, the role broadcasts the request to all actors under it; moreover, if a reply sent by an internal actor is not in majority, this reply will be delayed until further information can be obtained.

```
Role RW(){
 state {
    /*states for majority voting protocols*/
 }

 criteria {
   (a is an RW sensor) ∧ a.C(t)≠ 1;
 }

 policy {
  [request(msg)]{
     reroute(msg,*);//broadcast msg to all actors under the role
  }

  [reply(msg)]{
    if(msg is not in majority)
      delay(msg);
    else if(tell(msg.tag)=false)
      block(msg);
  }
 }
}
```

FIGURE 10 RW role.

A coordinator is needed to monitor and constrain message deliveries between the two roles in order to guarantee that the reply messages satisfy the timing constraint. Since the roles abstract the behaviors of actors and present a static interface to coordinators, the coordinators should see the behaviors of a group of actors instead of individual actor behaviors. Although different sensors differ in their accuracy and transmission time of data, the roles shield the sensor information from the coordinators and present to them only in group properties such as: once a request is sent to an actor below role IR, how long does it take for any actor below role IR to reply a valid message to the requesting actor.

To study the timing constraints between the two roles, we have to first study their individual timing behaviors. In the next subsection, we study the expected time for obtaining a valid vote using different voting protocols in open hostile environments. Curious readers of the ARC model could just take equation (13) and (22), leave the rest of the details and directly jump to Section 4.4 which describe the coordinator and coordination constraints. However, Section 4.3 offers a more in-depth explanation for the whole case study including the distributed voting problem.

The above design is concerned with hardware sensors. It should be noted that the general problem of surveillance and detection could be cast with software sensors. Take, for example, the problem of detecting an intruder in a network. This problem calls for distributing software components that act as sensors where each sensor may be different—designed to detect different forms of network attack—but their detection capabilities overlap. To reduce the problem of false alarms, the outputs of these sensors would be voted on to give assurance that an attack detection is valid.

4.3 Expected Time for Obtaining a Valid Vote in Different Voting Protocols

In this subsection, our discussion is based on the assumption that all the n sensor units provide datum D_i to the decision unit(s) and the inherently correct data value is D. The information credibility may not be at the fixed 100% level, that is, D_i may not always be the same as D. Instead, it may be time dependent. We use a credibility function $C_i(t)$ to describe the probability that D_i is the same as D at time t.

The following voting schemes are discussed here:

- ***1-out-of-n scheme.*** Under truthful assumption, we have that $D_i=D$, that is, every sensor unit provides correct data and $C_i(t)=1$. In this case, once the decision unit gets a datum D_i from any sensor, it can deliver D_i to the user without waiting for data from other sensors.
- ***k-out-of-n scheme.*** In the presence of faulty voters, a datum D_i given by a faulty voter may not be in agreement with the data of non-faulty voters. However, a datum D_i given by a non-faulty voter will be in close agreement with (or simply the same as) the data D of all the other non-faulty voters. We assume that the inherently correct data D is in the majority so that D can be determined by majority voting protocols. The credibility function $C_i(t)$ is given to be monotonic with bound of [0, 1]. The monotonicity indicates that with more time, we would get more trustworthy data.

We further assume that the probability distribution function for the time a sensor i takes to obtain and transmit data is given as $V_i(t)$. In other words, the probability that the decision unit get a datum from a sensor i by time t is given by $V_i(t)$.

To formulate the problem, let X_i be the random variable representing if the decision unit get a vote from the ith sensor

$$X_i = \begin{cases} 1, & \textit{if } \textbf{\textit{the vote of the i'th sensor is given}} \\ 0, & \textit{otherwise} \end{cases} \tag{5}$$

Thus, $P\{X_i=1\}=V_i(t)$, $P\{X_i=0\}=1-V_i(t)$. Moreover, we interpret data credibility as the probability that a given data D_i agrees with the inherent correct data D. Let Y_i be the random variable representing whether the data D_i agrees with D, that is

$$Y_i = \begin{cases} 1, & \textit{if } \textbf{\textit{the vote given by the i'th sensor is }} D \\ 0, & \textit{otherwise} \end{cases} \tag{6}$$

Thus, $P\{Y_i=1 \mid X_i=1\}=C_i(t)$, $P\{Y_i=0 \mid X_i=1\}=1-C_i(t)$. Therefore, the probability that the decision unit get a correct vote from the ith sensor is

$$p_i = P\{Y_i = 1 \cap X_i = 1\} = P\{Y_i = 1 \mid X_i = 1\} \times P\{X_i = 1\} = C_i(t)V_i(t) \tag{7}$$

and the probability that the decision unit cannot get a correct vote (either the vote is not given, or the given vote is incorrect) from the ith voter is

$$q_i = P\{Y_i = 0 \cup X_i = 0\} = P\{\overline{Y_i = 1 \cap X_i = 1}\} = 1 - p_i = 1 - C_i(t)V_i(t) \tag{8}$$

When all sensors are homogeneous, i.e., their $C_i(t)$ and $V_i(t)$ are identical, the probability that at least k similar (or the same as D) votes are collected is the summation of binomial distributions:

$$P\left\{\sum_{i=1}^{n}(X_i \wedge Y_i) \geq k\right\} = \sum_{i=k}^{n}\binom{n}{i}p^i(1-p)^{n-i} \quad \textit{where} \quad p = p_1 = \mathrm{L} = p_n = C(t)V(t) \tag{9}$$

Note that p is a function of t, it follows that equation (9) is the probability that at least k similar votes are collected before time t. Let random variable T represent the time point at which enough similar votes (at least k) are collected, i.e., the decision time, we have,

$$\begin{aligned} P\{T \le t\} &= \sum_{i=k}^{n} \binom{n}{i} p^i (1-p)^{n-i} \quad and \quad P\{T > t\} = 1 - \sum_{i=k}^{n} \binom{n}{i} p^i (1-p)^{n-i} \\ &= \sum_{i=0}^{k-1} \binom{n}{i} p^i (1-p)^{n-i} \end{aligned} \tag{10}$$

Therefore, the expected time that at least k same/similar votes are collected by the decision unit is

$$E[T] = \int_0^\infty P\{T > t\} dt = \int_0^\infty \sum_{i=0}^{k-1} \binom{n}{i} (C(t)V(t))^i (1 - C(t)V(t))^{n-i} dt \tag{11}$$

Note that in (11), different k's are used in distinct voting schemes. In *1-out-of-n* scheme where all sensors are truthful, we have that $k=1$. Whereas in *k-out-of-n* scheme, we have $k=\lceil (n+1)/2 \rceil$ in majority voting protocols and $k=\lceil 2n/3 \rceil$ in the more stringent Byzantine voting protocols. In the following subsections, we discuss these schemes separately, assuming $C(t)$ and $V(t)$ are given.

4.3.1 Truthful Voters. Under this scheme, we have $k=1$ and $C(t)=1$ in (11). We further assume that $V(t)$ is uniformly distributed over the interval $[0, T_1]$, i.e.,

$$V(t) = \begin{cases} \dfrac{t}{T_1}, & if\ t \in (0, T_1) \\ 1, & otherwise \end{cases} \tag{12}$$

Substitute k, $C(t)$, and $V(t)$ in (11), we have

$$E[T] = \int_0^{T_1} \left(1 - \frac{t}{T_1}\right)^n dt + \int_{T_1}^\infty (1-1)^n dt = \frac{1}{n+1} T_1 \tag{13}$$

Equation (13) indicates that as n increases, $E[T]$ decreases. In other words, under truthful assumption, resource availability positively impact data availability and system dependability. More careful observation reveals that the voting subsystem under truthful assumption is in fact a parallel system where the probability that the decision unit get at least one correct data from n sensors is

$$P\{\sum_{i=1}^{n} (X_i \wedge Y_i) \ge 1\} = 1 - P\{\sum_{i=1}^{n} (X_i \wedge Y_i) = 0\} = 1 - \prod_{i=1}^{n} q_i = 1 - \prod_{i=1}^{n} (1 - C_i(t)V_i(t)) \tag{14}$$

in which $\prod q_i$ characterizes a parallel system. In such a system, sensor units work in a "co-operative" way. Therefore, adding resources (more homogenous sensor units) to the subsystem improves its performance and thus reduces the expected decision time.

Similarly, consider a situation in which the data coming from the sensors are at constant rate (λ) for any unit interval, i.e., the number of data within a unit time is constant

over time. Based on the probability theory, we know that such event probability distribution can be modeled as exponential distribution, with the probability distribution function given below:

$$V(t) = 1 - e^{-\lambda t}, \quad t \geq 0 \tag{15}$$

Substitute k, $C(t)$, and $V(t)$ in (11), we have

$$E[T] = \int_0^{\infty} e^{-n\lambda t} dt = \frac{1}{\lambda} \cdot \frac{1}{n} \tag{16}$$

Therefore, though the probability distribution functions for voting time are different, if all the sensors are truthful, increasing n, i.e., the number of resources, reduces the expected time to obtain assured votes.

4.3.2 Untruthful Voters. Under untruthful voter scenario, we have k determined by the specific majority voting protocol (where $k=\lceil (n+1)/2 \rceil$ in majority voting protocols and $k=\lceil 2n/3 \rceil$ in the more stringent Byzantine voting protocols). We further assume that $C(t)$ is uniformly distributed over the interval $[0, T_2]$ and $V(t)=1$[1]. From (9), we can derive the probability of getting a valid data before time t:

$$P(t) = \sum_{i=\lceil (n+1)/2 \rceil}^{n} \binom{n}{i} \left(\frac{t}{T_2}\right)^i \left(1 - \frac{t}{T_2}\right)^{n-i} \quad (t \in [0, T_2]) \tag{17}$$

The following figures show the relationships between $P(t)$ and n under different t:

As can be seen that when $t=0.4T_2$, which means that the probability of getting a valid vote from an individual voter by time t is less than 50%, adding more homogeneously untruthful resources only makes it harder to get a consensus within given time. Intuitively, if over 50% chance a voter is to lie, adding more such voters only reduces the probability of getting valid votes within a given time. However, when $t=0.6T_2$, which means that the probability of getting a valid vote from an individual voter by time t is greater than 50%, adding more homogeneous resources facilitates the decision process, thus resulting in an increasing probability of obtaining a valid vote. The question now is: how does the resource availability influence the average decision time and thus the data availability?

Substitute $C(t)$, and $V(t)$ in (11), we have

$$\begin{aligned} E[T] &= \int_0^{T_2} \sum_{i=0}^{k-1} \binom{n}{i} \left(\frac{t}{T_2}\right)^i \left(1 - \frac{t}{T_2}\right)^{n-i} dt + \int_{T_2}^{\infty} \sum_{i=0}^{k-1} \binom{n}{i} (1)^i (1-1)^{n-i} dt \\ &= \sum_{i=0}^{k-1} \binom{n}{i} \int_0^{T_2} \left(\frac{t}{T_2}\right)^i \left(1 - \frac{t}{T_2}\right)^{n-i} dt \end{aligned} \tag{18}$$

[1]Although it is unreasonable to assume $V(t)=1$, i.e., a sensor is constantly giving out vote to the decision unit, we do this to simplify calculations and because not $V(t)$ alone but $C(t) \times V(t)$ characterizes the possibility that the decision unit gets a vote valued D, which is the inherently correct data.

Make the substitution $x=t/T_2 \Rightarrow dx=(1/T_2)dt$ in (18), we have,

$$E[T]=\sum_{i=0}^{k-1}\binom{n}{i}\int_0^1 x^i(1-x)^{n-i}T_2\,dx \tag{19}$$

Integrate by parts, we have,

$$\begin{aligned}\int_0^1 x^i(1-x)^{n-i}\,dx &= \frac{1}{i+1}\left[x^{i+1}(1-x)^{n-i}\Big|_{x=0}^{1}-\int_0^1 x^{i+1}d(1-x)^{n-i}\right]\\ &= \frac{n-i}{i+1}\int_0^1 x^{i+1}(1-x)^{n-i-1}\,dx\end{aligned} \tag{20}$$

Use mathematical induction on (20), we can prove that,

$$\int_0^1 x^i(1-x)^{n-i}\,dx=\frac{i!(n-i)!}{(n+1)!} \tag{21}$$

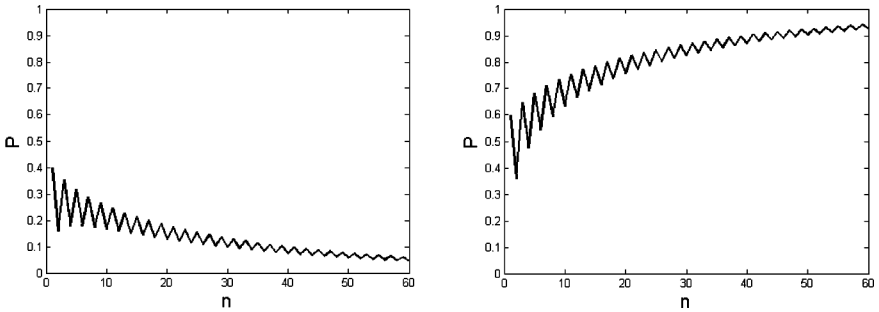
Therefore, from (19) and (21), we have that,

$$E[T]=T_2\sum_{i=0}^{k-1}\binom{n}{i}\frac{i!(n-i)!}{(n+1)!}=T_2\sum_{i=0}^{k-1}\frac{n!}{i!(n-i)!}\frac{i!(n-i)!}{(n+1)!}=T_2\sum_{i=0}^{k-1}\frac{1}{n+1}=\frac{k}{n+1}T_2 \tag{22}$$

Given that $k=\lceil (n+1)/2 \rceil$ and n is large, we have that $E[T]=T_2/2$

$$E[T]=T_2/2 \tag{23}$$

Therefore, in an openly hostile environment where not all voters are truthful, adding a homogeneous resource does not impact the expected time of getting a valid vote. The intuitive explanation for this result is that the integrated effects of (a) and (b) in Fig. 11 are neutralized.



(a) $t=0.4\ T_2$ (b) $t=0.6\ T_2$

FIGURE 11 The relationship between $P(t)$ and n.

Similarly, when the credibility function $C(t)$ is exponentially distributed on the interval $(0,\infty)$ with average rate λ, that is,

$$C(t) = 1 - e^{-\lambda t}, t \in (0,\infty) \tag{24}$$

Using equation (11), we have:

$$E[T] = \sum_{i=0}^{k-1} \binom{n}{i} \int_0^{\infty} (1 - e^{-\lambda t})^i (e^{-\lambda t})^{n-i} dt \tag{25}$$

Make the substitution where $x = e^{-\lambda t} \Rightarrow dx = -\lambda e^{-\lambda t} dt = -\lambda x dt$, we have,

$$E[T] = \sum_{i=0}^{k-1} \binom{n}{i} \int_1^0 (1-x)^i x^{n-i} \frac{1}{-\lambda x} dx = \frac{1}{\lambda} \sum_{i=0}^{k-1} \binom{n}{i} \int_0^1 (1-x)^i x^{n-i-1} dx \tag{26}$$

Integrate by parts and use mathematical induction, we can prove that,

$$\int_0^1 (1-x)^i x^{n-i-1} dx = \frac{i!(n-i-1)!}{n!} \tag{27}$$

Therefore, from (26) and (27), we have that,

$$E[T] = \frac{1}{\lambda} \sum_{i=0}^{k-1} \binom{n}{i} \frac{i!(n-i-1)!}{n!} = \frac{1}{\lambda} \sum_{i=0}^{k-1} \frac{n!}{i!(n-i)!} \frac{i!(n-i-1)!}{n!} = \frac{1}{\lambda} \sum_{i=0}^{k-1} \frac{1}{n-i} \tag{28}$$

where $k = \lceil (n+1)/2 \rceil$. The relationship between $E[T]$ and n in case of exponential distribution is illustrated in Fig. 12:

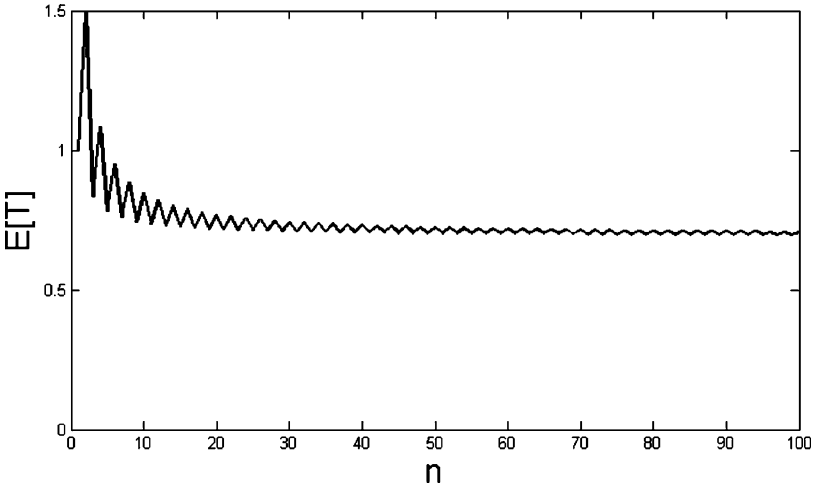


FIGURE 12 Expected Decision Time with λ=1.

As can be seen, when the number of working sensors are small, increasing the number of sensors generally decreases the expected decision time. However, since

$$\lim_{n\to\infty} \sum_{i=0}^{\lceil (n+1)/2 \rceil - 1} \frac{1}{n-i} = \ln n - \ln \frac{n}{2} = \ln 2 \approx 0.6931 \tag{29}$$

The expected decision time converges at $\ln 2/\lambda$ and no further decrease can be achieved by adding more resources. For example, with 11 sensors, the expected decision time is $0.7365/\lambda$, while with 23 sensors, the expected decision time is $0.7144/\lambda$ — a less than 3.0% time gain is at the cost of more than twice the resources.

4.4 Designing the Coordinators

Given the constraint that the time span between the delivery of the two data cannot be separated by a span larger than Δ, the coordinator could be declared as in Fig. 13.

Note that the timing constraint is expressed in the following format:

event1:event2 timeInterval

Such a constraint asserts that upon the occurrence of event event1, event event2 must happen within the time frame timeInterval. The timeInterval is a relative time duration: the starting point of such interval is relative to the time when event1 occurred. A timeInterval [d1, d2] is valid iff d1, d2 ∈ T, and d1 ≤ d2. Where T is a non-negative real number domain excluding spins.

Now, assume that the external actor sends the request at time t_0, and actor a_1 who is under role IR replies at t_1 and actor a_2 who is under role RW replies (the data is already checked for majority) at t_2. Given the assumptions and results in Section 4.3, together with the relative span requirement, we have the following timing constraints:

$$\begin{cases} t_1 - t_0 = \dfrac{1}{|A|+1} T_1 \pm \Delta_1 \\ t_2 - t_0 = \dfrac{k}{|R|+1} T_2 \pm \Delta_2 \\ |t_1 - t_2| \le \Delta \end{cases} \tag{30}$$

```
Coordinator C[IR r1,RW, r2](){
 state {}

 constraint {
   r1.tell():r2.tell()[0,Δ];
   r2.tell():r1.tell()[0,Δ];
 }

 policy {}
}
```

FIGURE 13 Coordinator.

where $|A|$ is the cardinality of the multicast declared in the states of role IR, $|R|$ is the number of actors under role RW, and k is the required majority. T_1 and T_2 are the corresponding parameters of the uniform distributions. Δ_1, Δ_2 are the possible deviations (e.g., standard deviations) from the expected decision times and Δ is the maximum time span of the two replies. $t_1-t_0=T_1/(|A|+1) \pm \Delta_1$ means $t_1-t_0 \in [T_1/(|A|+1) - \Delta_1, T_1/(|A|+1)+\Delta_1]$.

We further convert the constraints into the form $x_1-x_2 \le d$:

$$\begin{cases} t_1 - t_0 \le T_1/(|A|+1) + \Delta_1 \\ t_0 - t_1 \le -T_1/(|A|+1) + \Delta_1 \\ t_2 - t_0 \le kT_2/(|R|+1) + \Delta_2 \\ t_0 - t_2 \le -kT_2/(|R|+1) + \Delta_2 \\ t_1 - t_2 \le \Delta \\ t_2 - t_1 \le \Delta \end{cases} \tag{31}$$

Given such a set S of constraints of the form $x_j-x_i \le d_k$, we can construct a corresponding constraint graph $G=(V, E)$, where

$$E = \{(v_i, v_j) \mid x_j - x_i \le d_k \in S\} \text{ and } w(v_i, v_j) = d_k \tag{32}$$

The constraint graph instantiated from (31) is shown in Fig. 14.

With such a constraint graph, the Bellman-Ford algorithm may be used to detect if the graph has negative-weight cycles. A negative-weight cycle indicates that the constraints are infeasible. For example, the cycle will be $\overrightarrow{t_0t_1}, \overrightarrow{t_1t_2}, \overrightarrow{t_2t_0}$ negative if

$$\left(\frac{1}{|A|+1}T_1 + \Delta_1\right) + \Delta - \left(\frac{k}{|R|+1}T_2 - \Delta_2\right) < 0 \tag{33}$$

This means that role IR is so fast that role RW cannot catch up with it. In case of failure, the designers must reconsider the specification or declare exception handling codes. For example, when (33) occurs, the designer may slow down the reply from actors under role IR by:

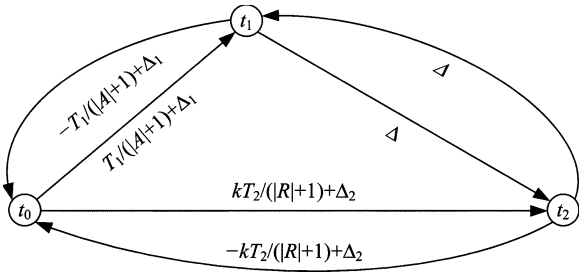


FIGURE 14 Timing constraint graph.

- delaying the delivery of a request to an actor under the role IR by changing the corresponding policy to

```
[request (msg)]{
   delay (msg,t)
   reroute (mdg, A);
}
```

- or changing the cardinality of multicast by removing some actor from the AidList declared in state.

5. Related Works

As argued by McDermott et al. [15], when considering the survivability of a system, we cannot assume that the system is susceptible to only one type of fault or another during the system's lifetime. Instead, a survivable and dependable system needs to not only correctly and accurately detect the presence of attacks or faults, but also function properly in the face of these faults, especially in mission-critical systems. At the same time, these mission critical systems should also be able to survive faults that are random and unpredictable in nature. Unfortunately, it is still a common practice in security and fault-tolerance community that an ad hoc-chosen, isolated, independent security mechanism, such as firewalls, intrusion detection systems, and software sandboxes is used in building a survivable system.

Recent research has started focusing on integrating different security techniques to block, evade, and react to attacks to improve system survivability. Keromytis's SABER (Survivability Architecture: Block, Evade, React) [9] is an example of the new effort. The SABER architecture uses a coordinated multi-layer approach to defend against attacks targeted at different levels of the network stack. The coordination and correlation of different layers are through a publish-subscribe, event-based infrastructure built into the model. The similarity between SABER and ARC lies in that they both use a multi-layered approach and use light-weighted events as the coordination media. The major difference is that in the SABER model, the coordination and the workflow among different approaches and different layers are built into the framework and it is difficult to customize for different applications—requiring users to understand the framework. In contrast, in the ARC model, both the coordinators and roles are not only proactive entities at runtime, but they also offer high-level user programmable language constructs—allowing a user to easily deploy the ARC into different applications with different attack protection workflows.

Brian Robinson [20] proposed a three-layered protection model that can be mapped to the ARC model. More specifically, a well-protected enterprise will require the following elements that are all working in concert with one another:

- Data-level protection is handled at the actor level. The semantics of this level security is to protect its internal data. As actors communicate with the outsiders only through messages, the actors may decide to encrypt messages they send out; filter out or preprocess messages they decide to receive and hence protect their data integrity.
- Application-level protection is handled by roles. The roles not only define the properties and behaviors that actors (the active processes) by presuming the role they must take, but they also serve as functionality boundaries.
- Network-level protection is achieved through the collaboration of actors, roles, and coordinators. If the actors are well protected, the only breach left for the attackers are messages on the network. The attackers may remove messages from the network

and hence prevent them from being processed by the receiving actors. (The modification of messages is handled at the actor level.) However, if an appropriately observing mechanism is built within the roles and the coordinators, such message loss should be able to be detected by the coordinators.

In our ARC model, we consider coordination as solutions to not only the general dependability problem but to obtaining self-adaptiveness for survivability. In the coordination literature, coordination models and languages can be classified as either data-oriented or control-oriented. For instance, the Tuple Space model and its corresponding language Linda [7] uses a data-oriented coordination model, whereas the IWIM model and corresponding language MANIFOLD [2] is based on a control-oriented coordination. The activity in a data-oriented application tends to center around a substantial body of shared data; the application is essentially concerned with what happens to the data. On the other hand, the activity in a control-oriented application tends to center around the processing or the flow of control; such an application is essentially described as a collection of activities that genuinely consume their input data, and subsequently produce, remember, and transform this data, thus generating "new data" by themselves.

The Linda-like coordination tends to lack flexibility and control both in agent-to-agent coordination and accesses to local data. For instance, local data are bound by the built-in data-access primitives, such as write, read, read-with-remove. Any coordination policy not directly supported by the model typically falls upon the responsibility of functional entities [3]. In open, heterogeneous and unreliable environments, proposals have been made to enhance the Linda communication kernel either by

1. adding new general purpose primitives or
2. by making it programmable.

The former approach has been adopted by many researchers, such as [24], to add access control primitives for supporting a more secure tuple space, and [8] that creates an ATSpace to support an application-oriented broker service for the tuple space. However, adding new operations to a shared information space tends to limit the uncoupling of the coordination model from the application because the applications must be made aware of what the admissible operations are in order to effectively interact with each other. The latter approach, without adding new admissible operations, makes it possible to program the behavior of the communication abstractions in response to communication events. The tuple center model [16] is one such approach. The tuple center model decouples coordination from computation by separating individual perceptions of coordination and the global coordination issues (such as, constraints, rules). A language called ReSpecT specifies such rules and embeds the control of the rules within the tuple space. The model avoids fixed primitives; instead, it allows a user programmable coordination control and hence is more flexible.

There are a few coordination models that are based specifically on the actor models, such as Frolund's Synchronizers [6, 5] which separates local synchronization constraints from multi-actors, and Varela's directors [23] which uses a hierarchical structure to encapsulate coordination among a cast of actors. However, the directors are not transparent to the actors and before a message is being received, a hierarchical constraint checking must be done requiring each actor and director to know its directors in the hierarchy.

The ARC model integrates the advantages of data -oriented and control-oriented coordination approaches. The coordination data are actor messages; however, the coordination controls are distributed into roles and coordinators. Roles and coordinators can mutually provide some level of security checking and provide the system with increased survivability.

6. Conclusion

As we have observed over the past decade, critical information systems can no longer be encapsulated inside ivory towers, instead, not only do they reside in an open environment, but also the structure of the system itself becomes distributed and the integrity of the system depends on the coordination of distributed composing elements.

The open environment opens more potential vulnerabilities to attackers and hence calls upon new methodologies and technologies to counter-attack and protect the systems. The ARC coordination model is presented as an attempt to use coordination strategies to improve system survivability. The ARC model distributes coordination constraints, especially fault-tolerance polices, among roles and coordinators. From the design point of view, the model not only avoids a single point of coordination failure in an openly hostile environment, but also provides two survivable feedback loops to detect and contain possible attacks and transform fault-tolerance to attack-tolerance.

Moreover, in an open environment, the number of functional entities (i.e., actors), can be very large and quite dynamic; yet the functional categories of the actors of a system, (i.e., the roles that actors play), are relatively small in number, and stable. Hence, from a coordination perspective, role-based coordination is lightweight and more scalable compared with individual, functional, entity-based coordination, especially in open systems. Furthermore, as the roles shield the dynamics of the actors, the logic of role-based coordination becomes simpler and more reusable.

The notion of defense in-depth motivates us to ask: can the ARC model be applied to cyber defenses themselves? Firewalls, anti-virus software, and intrusion detection systems act as preventive attack measures; yet these very measures are frequently targeted by attackers whose aim is to first disable these defenses so that future attacks go undetected. We seek to find how the ARC model can be applied to attack-detectors. By defining these defenses as actors who carry out defensive roles, the ARC's feedback mechanism could link these defenses together into a self-checking alarm system.

Acknowledgement

This research is supported by NSF under grant CNS 0431832.

About the Authors

Dr. Shangping Ren is an Assistant Professor in the Department of Computer Science at the Illinois Institute of Technology. Her research interests include real-time embedded systems, coordination modeling, and programming languages. Currently she leads two projects founded by the National Science Foundation and Fermi National Accelerator Laboratory focused on open distributed and embedded systems.

Yue Yu received the B.S. degree in Computer Software in 2005 from Tsinghua University, China. He is currently a Ph.D. student in the Computer Science Department at Illinois Institute of Technology. His research interests include real-time embedded systems and coordination model for distributed systems.

Kevin A. Kwiat has been a civilian employee with the U.S. Air Force Research Laboratory in Rome, New York for over 23 years. He received the BS in Computer Science, the BA in Mathematics, the MS in Computer Engineering, and the Ph.D. in Computer Engineering, all from Syracuse University. He holds 3 patents. In addition to his duties with the Air Force, he is an adjunct professor of Computer Science at the State University of New York at Utica/Rome, an adjunct instructor of Computer Engineering at Syracuse University, and

a Research Associate Professor with the University at Buffalo. He recently completed assignments as an adjunct professor at Utica College of Syracuse University, a lecturer at Hamilton College, and a visiting scientist at Cornell University. His main research interest is dependable computer design.

Jeffrey J. P. Tsai received a Ph.D. degree in Computer Science from Northwestern University, Evanston, Illinois. He is a Professor in the Department of Computer Science at the University of Illinois at Chicago where he is also the Director of the Distributed Real-Time Intelligent Systems Laboratory. He was a Visiting Computer Scientist in the U.S. Air Force Rome Laboratory and is a Senior Research Fellow of IC2 at the University of Texas at Austin. He is an author or co-author of 10 books and over 200 publications in the areas of knowledge-based software engineering, software architecture, formal modeling and verification, distributed real-time systems, sensor networks, ubiquitous computing, trustworthy computing, intrusion detection, software reliability and adaptation, multimedia systems, and intelligent agents. He has chaired or co-chaired over 20 international conferences and serves as an editor for 9 international journals. He received a University Scholar Award from the University of Illinois Foundation, an IEEE Meritorious Service Award from the IEEE Computer Society, and an IEEE Technical Achievement Award from the IEEE Computer Society. He is a Fellow of the IEEE, the AAAS, and the SDPS.

References

1. G. Agha, *Actors: A Model of Concurrent Computation in Distributed Systems*, MIT Press, 1986.
2. F. Arbab, "IWIM: A communication model for cooperative systems," In *Proceedings of the 2nd International Conference on the Design of Cooperative Systems*, pp. 567–585, 1996.
3. M. Cremonini, A. Omicini, and F. Zambonelli, "Coordination and access control in open distributed agent system: The tucson approach," In *A. Porto and G. C. Roman, editors, Coordination Languages and Models*, volume 1906, pp. 99–114. LNCS, Springer-Verlag, 2000.
4. A. Degani, *Taming HAL: Designing Interfaces Beyond 2001*. Palgrave Macmillan, 2003.
5. S. Frølund, *Coordinating Distributed Objects: An Actor-Based Approach to Synchronization*. MIT Press, 1996.
6. S. Frølund and G. Agha, "Abstracting interactions based on message sets," In *Object-Based Models and Languages for Concurrent Systems*, Lecture Notes in Computer Science, *vol. 924, pp 107–124, Springer Verlag*, 1996.
7. D. Gelernter, "Generative communication in Linda," In *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7, 80–112, 1985.
8. M.-W. Jang, A. Ahmed, and G. Agha, "A flexible coordination framework for application-oriented matchmaking and brokering services," in *Technical Report UIUCDCS-R-2004-2430, Department of Computer Science, University of Illinois at Urbana-Champaign*, 2004.
9. A. D. Keromytis, J. Parekh, P. N. Gross, G. Kaiser, V. Misra, J. Nieh, D. Rubenstein, and S. Stolfo, "A holistic approach to service survivability," in *SSRS '03: Proceedings of the 2003 ACM workshop on Survivable and self-regenerative systems*, New York, NY, USA, 2003. ACM Press, pp 11–22.
10. K. S. Killourhy, R. A. Maxion, and K. M, "Tan. A defense-centric taxonomy based on attack manifestations," In *International Conference on Dependable Systems & Networks*. IEEE, 2004.
11. K. Kwiat, K. Ravindran, and P. Hurley, "Energy-efficient replica voting mechanisms for secure real-time embedded systems," In *Proceedings of the Sixth IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, Taormina – Giardini Naxos, June 2005.
12. J.-C. Laprie, "Dependability of software-based critical systems," In *Dependable Network Computing*. Kluwer Academic Publishers, 1999.
13. E. A. Lee, "Building unreliable systems out of reliable components: the real time story," *Abstract of Invited Plenary Talk, Monterey Workshop, Laguna Beach, California*, September 22, 2005.

14. P. Lee and T. Anderson, *Fault Tolerance: Principles and Practice, 2nd Edition, Dependable Computing and Fault-Tolerant System*, volume 3. Springer-Verlag, 1990.
15. J. McDermott, A. Kim, and J. Froscher, "Merging paradigms of survivability and security: stochastic faults and designed faults," in *NSPW '03: Proceedings of the 2003 workshop on New security paradigms*, New York, NY, USA, 2003. ACM Press, pp. 19–25.
16. A. Omicini and E. Denti, "From tuple spaces to tuple centers," *Science of Computer Programming*, **41** 3, 277–294, 2001.
17. B. Randell, "Facing up to faults," *The Computer Jorunal*, **43** 2, 95–106, 2000. Turing Memorial Lecture.
18. S. Ren, N. Chen, Y. Yu, P.-E. Poirot, L. Shen, and K. Marth, "Actors, roles and coordinators a coordination model for open distributed embedded systems," In *COORDINATION 2006*, pp 247–265, 2006.
19. R. J. Ellison, D. Fisher, R. Linger, H. Lipson, T. Longstaff, and N. Mead, "Survivable network systems: an emerging discipline," Technical Report CMU/SEI-97-TR-013 and ESC-TR-97-013, Carnegie Mellon University, Software Engineering Institute, Nov. 1997, Rev. May 1999.
20. B. Robinson, "Data protection: Emerging threats spur technology updates and a new class of security devices," *Federal Computer Week*, **19**, 35, Oct. 2005.
21. C. M. U. Software Engineering Institute. Survivable network analysis. http://www.sei.cmu.edu/programs/nss/.
22. J. P. G. Sterbenz, R. Krishnan, R. R. Hain, A. W. Jackson, D. Levin, R. Ramanathan, and J. Zao, "Survivable mobile wireless networks: issues, challenges, and research directions," in *WiSE '02: Proceedings of the 3rd ACM Workshop On Wireless security*, pages 31–40, New York, NY, USA, 2002. ACM Press.
23. C. A. Varela and G. Agha, "A hierarchical model for coordination of concurrent activities," in *International Conference on Coordination (COORDINATION '99), LNCS 1594, pp 166-, 1999*.
24. A. Wood, "Coordination with attributes," in *Proceedings of the 3rd International Conference on Coordination Models and Languages*, 1999.
25. K.-P. Yee, "User interaction design for secure systems," in *Proceedings of the International Conference on Information and Communication Security*, 2002.
26. R. Ziaei and G. Agha, "SynchNet: a petri net based coordination language for distributed objects," in *Proceedings of the second international conference on Generative programming and component engineering*, pp. 324–343, New York, NY, USA, 2003. Springer-Verlag New York, Inc.