Bridging the HLA: A Case Study in Composing Publish-Subscribe Systems[*]

J. Dingel
Queen's University
Kingston, Ontario K7L 2Y1
dingel@cs.queensu.ca

D. Garlan
Carnegie Mellon University
Pittsburgh, PA 15213
garlan@cs.cmu.edu

C. Damon
University of Vermont
Burlington, VT 05405
cdamon@cs.uvm.edu

ABSTRACT

The recent popularity of publish-subscribe (pub-sub) system architectures has led to a desire for a refined architecture that supports the composition of pub-sub systems. One proposed solution links such systems using a special bridge component that acts as a mediator, passing events between the systems. The bridge appears to be an ordinary pub-sub component to each system, effectively serving as a surrogate for the other system. Although attractive as a lightweight pub-sub combinator, the notion of a pub-sub bridge raises a number of questions, such as whether its use introduces new sources of deadlock or inconsistency. In this paper, we show that designing such a bridge is far from trivial, and indeed requires special treatment to achieve desired properties. To make these issues concrete, we describe our results in analyzing the feasibility of a bridge for the HLA, a standardized pub-sub framework designed for distributed simulation applications. We identify a small set of core problem classes for pub-sub bridge designs. Additionally, we also classify a set of generic solution paths and show how each applies to the problem classes. Although based on the HLA, we believe that these problems and solutions are applicable to many pub-sub systems as well as to other architectures for loosely coupled distributed systems.

Keywords

Software architecture, publish-subscribe systems, High-Level Architecture, bridge component, component integration, distributed systems

[*]This research was sponsored in part by the Defense Modelling Simulation Office (DMSO), by the Department of Defense (DOD), the Army Research Office (ARO) under Contract No. DAAD19-01-1-0485, by the Defense Advanced Research Projects Agency (DARPA) under Contracts No. F30602-00-2-0616 and N66001-99-2-8918, and by the Air Force Research Laboratory under Contract No. F30602-00-C-0000.
The views, findings and conclusions or recommendations contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the DMSO, DOD, ARO, DARPA, or the United States Air Force.

Submitted for Publication.

1. INTRODUCTION

Publish-subscribe (pub-sub) systems, implemented as collections (or federations) of loosely coupled components which communicate through multicast, are becoming an increasingly important style of system architecture. The components in a pub-sub system publish information through events to which other components can subscribe. Components can be added or removed to a pub-sub system without the direct knowledge or cooperation of the other components. This decoupling of components within a pub-sub federation in principle makes such systems particularly adaptable. As a consequence, pub-sub mechanisms can be found in a wide spectrum of today's systems including automotive control, programming environments, user interface frameworks, space software, service location and discovery systems (e.g., Jini [10]), and component integration standards (such as JavaBeans [12], or CORBA [4]).

In addition to providing basic event dispatch mechanisms, the infrastructure for pub-sub systems typically provides a number of other critical services. Commonly supported services include

- the orderly starting and stopping of a federation (e.g., to ensure that events are not lost due to start-up race conditions),
- membership management (e.g., to determine which components are part of a federation),
- data management services (e.g., to determine which components are affected by changes to shared state), and
- timing services (e.g., to enforce causal ordering of event delivery).

Driven by the prevalence of pub-sub systems, there has been considerable recent interest in finding lightweight mechanisms to compose multiple federations. Ideally, such a composite federation would permit separately-developed, and separately-specified federations to work together, without significant modification to any of the individual federations or to their run-time infrastructure. Moreover, suitable glue mechanisms could provide a mechanism for limiting the visibility of information between federations. For example, a federation might limit the kind or precision of data that is exported to other federations.

In realizing such a scheme, how the various federations might be linked becomes an important question. One previously proposed solution provides the glue in the form of a special bridge component that links two federations. In this solution, the bridge acts as a mediator, passing events between the two federations. The bridge would appear to be an ordinary pub-sub component to each federation, effectively serving as a surrogate for the entire other federation.

The use of a bridge component is architecturally attractive for a number of reasons. The bridge simply looks like any other pub-sub component, so multiple federations can be joined transparently to the joined federations. In principle, the use of a bridge should require no changes to the existing pub-sub infrastructure: by using existing services and event subscriptions a bridge should be able to update each side appropriately. Furthermore the bridge could handle any filtering or event translation needed to match impedances of the joined federations or to enforce security restrictions.

Although attractive in principle, the notion of a pub-sub bridge raises a number of questions. Does the introduction of a bridge introduce sources of deadlock or inconsistency? Can a bridge obtain enough information from each federation to keep the sides in sync? Are there special protocols of interaction that a bridge developer should be aware of to ensure that the bridge is behaves correctly?

In this paper, we show that designing such a bridge is far from trivial. While the simple transmission of events is straightforward, the inclusion of other essential services (such as those mentioned above) is considerably more complex, and indeed requires special treatment to achieve desired properties. To make these issues concrete, we describe our results in analyzing the feasibility of a bridge for the High-Level Architecture (HLA) [13], a standardized pub-sub framework designed for distributed simulation applications. Specifically, we show that it is possible to identify a small set of core problem classes for such pub-sub bridge designs. Additionally, we also classify a set of generic solution paths and show how each applies to the problem classes. While based on the HLA, we believe that the problems and solutions are applicable to many pub-sub systems, and, indeed, other architectures for loosely coupled distributed systems as well.

2. RELATED WORK

Three general areas of design work relate to this paper: distributed systems, publish-subscribe systems, and the HLA. In the area of distributed systems, researchers have investigated algorithms for coordinating loosely-coupled distributed processes. These algorithms address the problem of achieving consensus about various system properties, such as membership, time, and topology (e.g., [14, 5]). While this research is relevant in identifying the kind of algorithms that can lead to inconsistency or potential deadlock, it does not directly address the problem of bridging collections of loosely-coupled components with lightweight mechanisms such as the a pub-sub bridge.

Many researchers have investigated publish-subscribe systems directly, both from an engineering perspective (e.g., [15], [2]), and from a foundational perspective (e.g., [9, 1, 8]). Most of these treatments have been concerned with the problems of building or reasoning about single pub-sub federations. In contrast, our research looks at composing *multiple* federations. One exception is the C2 system [16] which has an architecture consisting of multiple pub-sub connectors, arranged hierarchically. In that work, pub-sub connectors may be joined together directly or via a component, thereby providing the potential for lightweight pub-sub bridging mechanisms similar to the one we investigate here. However, C2 research has not focused as much on algorithms and protocols for maintaining global forms of consistency, or the impact of a bridge combinator on such algorithms.

The HLA itself has generated considerable attention from the practitioner community who uses it [11]. Most of this work has focused on the properties of a single HLA and other similar integration frameworks, since the bridge is still a fairly new proposal. Earlier work of by one of the authors of this paper investigated the formal specification and analysis of the HLA from an architectural perspective [1], but did not look at all at bridging issues.

3. THE HIGH-LEVEL ARCHITECTURE

The High Level Architecture (HLA) for distributed simulation defines a framework for the integration of cooperating, distributed simulations, possibly built by many vendors [13]. Initially proposed by the Defense Modeling and Simulation Office, it is now an IEEE Standard and widely used in practice [3].

The HLA defines a distributed simulation as a collection (called a *federation*) of semi-independent simulations (each called a *federate*) that communicate using the services provided by a run-time communications infrastructure (called an *RTI*). Simulation events are communicated using a pub-sub model: new values of simulated entities announced by one federate will be received by the federates who subscribe to those events. Events are characterized in terms of updates to attributes of objects that are defined by a shared object model (called the *Federation Object Model*, or *FOM*). In addition, the RTI provides a large number of services to handle other mechanisms for coordinating simulations. These services are grouped into five different categories and include support for

- starting and stopping a federation, synchronizing federates, and saving and restoring federation state (Federation Management Services),
- declaring potential ownership of or interest in certain objects (Declaration Management Services)
- updating and reflecting object attributes, creating and deleting objects (Object Management Services)
- transferring object ownership from one federate to another (Ownership Management Services), and
- global clock management (Time Management Services)

In its original design, the HLA assumed that a federation would be composed of a single RTI coordinating a single set of federates. More recently, however, there has been considerable interest in being able to define a federation as a set of linked RTIs each with their own sets of federates. Such a composite federation allows two or more independently-developed federations to work together, without requiring significant modification to the individual federations or to their RTIs. Additionally, suitable glue mechanisms could act as a filter, transforming or hiding certain events between federations.

One proposed solution to realize such a composition is to use a bridge federate to link two federations [13]. The bridge would appear as an ordinary federate to each federation, ideally requiring no changes to the RTI. The bridge might implement various filtering and transformation policies on events, but it should encapsulate those policies, making them easy to change. Moreover, the intention was for the bridge to be a relatively simple component: for example, it would not be appropriate for a bridge to, for instance, maintain an amount of state similar that maintained by the RTI. To achieve this transparency, seamlessness and simplicity, the bridge was conceptualized as consisting of three logical parts, illustrated in Figure 1. As illustrated, bridge B joins the two federations F and G. The bridge B itself consists of three parts:

Surrogate s_F: Federate that interacts with the federation G on behalf of the federation F. We say that the surrogate s_F *represents* the federation F. s_F reflects relevant properties of the federation it represents to its federation, that is, the federation it is connected to. Note F may be connected to more federations through a second bridge. Intuitively, s_F represents all federations to the “left” of the bridge B.

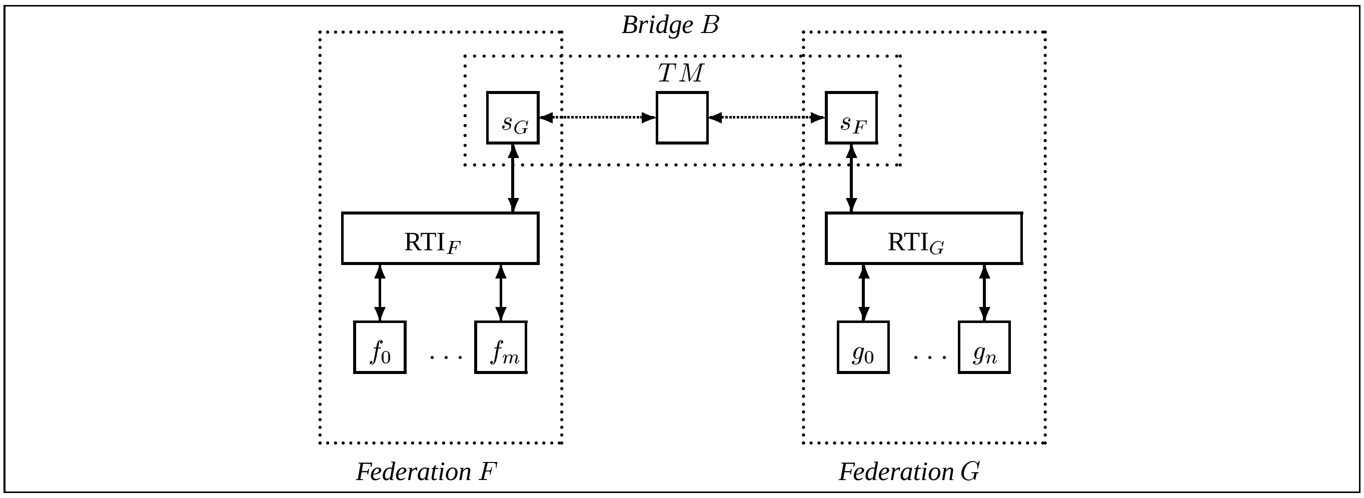

Figure 1: Federations F and G connected by a bridge B

Surrogate s_G: As above, except that every occurrence of s_F, F, and G is replaced by s_G, G, and F respectively. Intuitively, s_G represents all federations to the "right" of the bridge.

Transformation manager TM: Module that translates between the two FOMs through a mapping that associates an entity (e.g., service, object, attribute, interaction) from one side of the bridge with corresponding entities on the other side of the bridge. Possibly carries out additional transformations, and thus may function as a guard.

Using several bridges, federations could theoretically be linked to form linear, hierarchical, or graph-like topologies. However, earlier work has already identified some serious problems with circular structures [6]. For instance, if bridges connect federations in a circular fashion and no special provisions are taken, the invocation of a service may give rise to an infinite number of invocations of the same service with each new invocation overriding the old one.

¿From a semantic point of view a bridge should have two key properties:

1. It should respect RTI semantics: The behavior of a bridged system should be the same as the behavior of a corresponding unbridged system with all of the federates linked by a single RTI, modulo naming and filtering issues. Consider, for instance, the bridged system in Figure 2. Consider federate f_0. The bridge B_1 should create the illusion to f_0 that its federation F is joined not only by f_1 but also by a federate whose properties are given by the sum of the properties of the federates g_0, g_1, and h_0. In other words, to each of the federates f in federation F, the existence of the bridge should be noticeable to f only in so far it makes the attributes of the federates g_0, g_1, and h_0, available to f. Note that according to the current standard [3], a federate cannot determine the number or identity of federates that participate in its federation. Figure 3 attempts to illustrate the effect of the bridges in Figure 2 from the federates' perspective.

2. It should respect federate semantics: the behavior of a bridge should be identical to any other federate, that is, its behavior should be a subset of the behavior that a normal federate might exhibit. This implies, for instance, that the bridge surrogates cannot exhibit behaviour that a normal federate could not exhibit.

However, as noted earlier, it is not apparent that such a bridge can be built to satisfy these properties without significant modification to the existing HLA standard given in [3]. In the remainder of this paper, we describe the results of our investigation of problems that arise when using a bridge. For the purposes of our analysis we consider the case of linear topologies of federations.

In carrying out this work, our approach was to use the HLA standard specification [3] (and to some extent a previously formalized model [6]) to look for anomalous situations. We then categorized those situations in the form of more general problem classes. In this way, we are able to identify problem areas that both capture a large number of parts of the HLA and extend beyond the specific protocols of the HLA. Having identified problem classes, we then attempted to classify possible solution paths, and to understand the mapping between problems and solutions. For more details on this work, the reader is asked to refer to the technical report [7].

The next section will present an HLA service in more detail. Discussion of the implementation of the service in the presence of a bridge will bring us to our first problem category. Section 5 presents the other problem categories while Section 6 discusses the solution categories.

4. AN EXAMPLE PROBLEM

We now illustrate the kind of problems that arise when trying to extend an HLA service to the bridged case.

4.1 Federation Save

In the HLA, any federate may make a request to the RTI that all federates in the federation checkpoint their state. Such checkpoints can be used to recover from federation failures, by restoring a federation to a previously well-defined state.

In the unbridged case, invocation of federation save causes the following protocol to be executed:

1. To request a save, a federate invokes the *Request Federation Save* service on the RTI.

2. The RTI responds by sending the *Initiate Federate Save* †[1] to

[1] In the HLA, events announced by the RTI are decorated with a

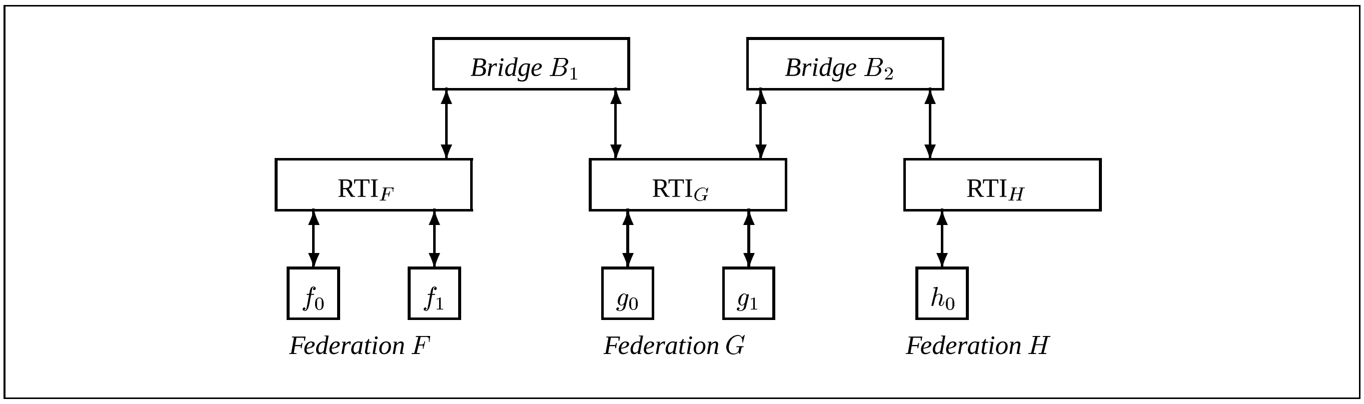

Figure 2: Three federations linked by two bridges

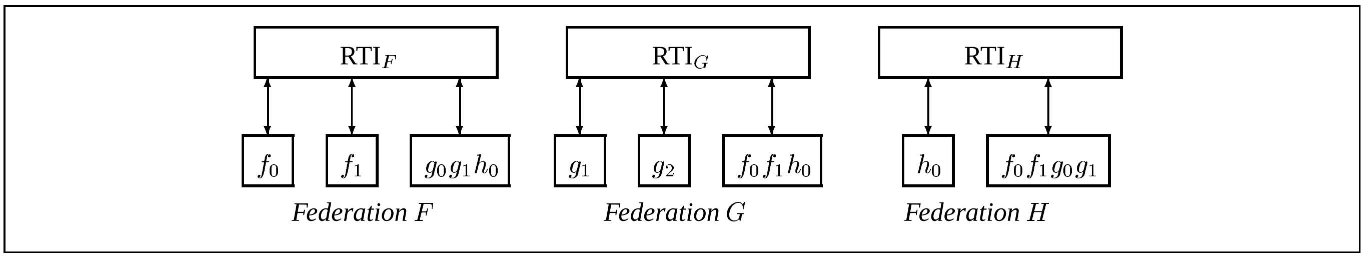

Figure 3: The effect of the bridges in Figure 2 from the federates' perspective

every federate in the federation.

3. As soon as a individual federate has completed saving its own state, it invokes the *Federate Save Completed* service.

4. Once every federate has informed the RTI that its save has completed, the RTI announces the completion of the save to each federate using the *Federation Saved* † service.

Figure 4 shows the message sequence chart corresponding to the above protocol for a federation F containing three federates f_0, f_1, and f_2.

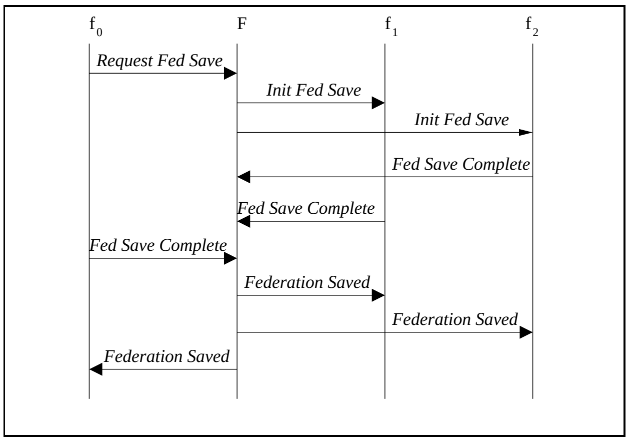

Figure 4: Example of save protocol for unbridged case

dagger †.

4.2 Bridging the Protocol

To extend the above protocol to the bridged case, the bridge must ensure that a surrogate

- requests a save on behalf of the federation it represents, that is, whenever a save request was issued in the federation it represents, the surrogate must issue a save request in the federation it is connected to.

- reflects the successful completion of the request, that is, whenever the federation it represents has saved successfully, it must announce the completion in the federation it is connected to.

In other words, the bridge must propagate two kinds of information: the initial request for the save and the completion of the save in each federation being bridged. We represent this communication as, respectively, *Request Save* and *Save Completed* messages sent across the bridge.

Assuming that the bridge serves simply transmits initiation and completion of saves, we then obtain the following protocol:

1. To request a save, a federate invokes the *Request Federation Save* service on the RTI.

2. The RTI responds by sending the *Initiate Federate Save* † to every federate and surrogate in the federation.

3. When a surrogate receives the notification to initiate a save from the RTI, it sends a *Request Save* message to the surrogate at the other end of the bridge.

4. When a surrogate receives a *Request Save*, it sends the *Request Federation Save* service to its RTI.

5. As soon as a federate has completed saving its own state, it invokes the *Federate Save Completed* service.

6. When all federates that a surrogate represents have completed the save, the surrogate issues a *Federate Save Completed*.

7. Once every federate has informed the RTI that its save has completed, the RTI announces the completion of the save to each federate using the *Federation Saved* † service.

Figure 5 shows the message sequence chart corresponding to the above protocol for a bridge that connects two federations F (containing f_0, f_1, and s_G) and G (containing g_0, g_1, and s_F).

4.3 Deadlock!

On the surface the protocol above seems like a natural extension to the bridged case: in this case the bridge simply triggers saves on the other side, and notifies the initiating RTI when it is saved. This behavior has the desirable properties indicated in Section 3 of making surrogates surrogate behavior indistinguishable from ordinary federates, and of keeping the bridge simple. It also requires no new additions to HLA protocol, since the only new activity is encapsulated within the bridge itself (i.e., between the its two surrogates).

Unfortunately this protocol deadlocks. This is because Surrogate s_F cannot send the *Save Completed* message across the bridge until it knows that all other federates in federation G have completed saving. But because the surrogate is itself viewed by the RTI as one of the federates of G, it will never get the *Federation Save Completed* † message from the RTI of G because it will not receive the *Save Completed* message from s_G. In other words, for s_F to be able to send out a *Save Completed* to s_G it must have received a *Save Completed* from s_G. Similarly, for s_G.

As we will see, there are several possible ways that one might handle the situation. One is to break the circularity by allowing each surrogate to send *Save Completed* as soon as *all federates in its federation except itself* have saved. However, according to the HLA standard federates do not have the capability to obtain this information. We thus must give the surrogates the additional capability to determine when all federates in its federation except itself have saved successfully. This can only be done by changing the set of services and events of HLA itself – a change we would prefer not to make.

4.4 The Consensus Problem Category

Although couched in terms of a particular protocol for federation save, the problem exposed by the above analysis is not unique to that protocol. Indeed, there are a number of places where the same problem can arise in pub-sub frameworks such as the HLA. More precisely, in the HLA the consensus problem occurs not only in the context of the save service, but also in, for instance, the protocols for synchronization and time advance:

- Synchronization points are provided by the infrastructure to allow federates to coordinate along policy-defined lines. A common usage of a synchronization point is to announce the completion of initialization processing. Any federate may register a synchronization point. If the registration is accepted, the RTI informs each federate of the synchronization point. As soon as each federate has completed the expected work, it announces that it has achieved the synchronization point. Once all federates have achieved the synchronization point, the RTI announces that synchronization has been achieved.

- Logical time logical time can advance for each federate only when it can advance for all federates. In the time advance protocol, each federate announces how far it is prepared to advance time using the *Time Advance Request* or *Time Advance Request Available* service. Each of these services includes a parameter indicating how far the federate is willing to have time advance. By invoking these services, the federate is guaranteeing that it will not send any more time stamped messages with a time stamp prior to the indicated time (or prior or equal to the indicated time for *Time Advance Request Available*). When every federate has agreed to advance beyond the requested time for any federate, the RTI responds by sending the *Time Advance Grant* † service to the federate. The federate may then advance its logical time to the time given in the *Time Advance Grant* † service.

Just like save, synchronization and time advancement require consensus between federates. Therefore, the naive adaptations of these two protocols to the bridged case suffer from the same problem as the save protocol described above and will therefore deadlock (an example of the bridged protocol for synchronization is shown in Figure 6). As for save, enriching the capability of the surrogate breaks the circularity, but also stands in conflict with the HLA standard.

In order to capture the general case, we can generalize the problem by considering it as one that is triggered whenever a federation of pub-sub components must reach consensus. For the HLA, if surrogates are restricted to federate capabilities, the consensus decisions of each surrogate of a bridge are mutually dependent. Circularity is broken by giving the surrogate the capability to make the consensus decision by considering the state of the federates in its federation alone. But this in turn requires making changes to the protocols for the non-bridged case.

5. OTHER PROBLEM CATEGORIES

Consensus is not the only category of problem. Through our investigation of the HLA we have been able to identify five other problem categories, which we detail in this section.

5.1 Service Barriers

The consensus problem induces a second category of problems that may not, at first, be obvious. The problem arises from the fact that one federation must always announce consensus first. Federates within that federation may choose to perform actions only allowable, by protocol or by policy, after consensus has been achieved. The bridge may forward these actions to a second federation that has yet to announce consensus. A critical race thus ensues: if that action is propagated before consensus is announced, a disallowed action will be taken. We term problems of this nature *service barrier* problems.

To illustrate the problem in the HLA, we return to the synchronization service and its underlying protocol already described above and shown in Figure 6. Apart from the consensus problem, we run into a second problem when trying to bridge the synchronization protocol. Suppose, for instance, that a federate requests a synchronization by registering a synchronization point with its RTI. In an unbridged federation, the RTI implementation may be such that every synchronization request has to be completed before another one can be registered. More precisely, once a federate f has been informed of the synchronization point by the RTI, f may assume that it will not receive another announcement of a different synchronization point until this request has been serviced.

In the bridged case, however, it is unreasonable to assume that all federates notify their RTIs of successful synchronization at exactly the same time. Consequently, some federates may receive the

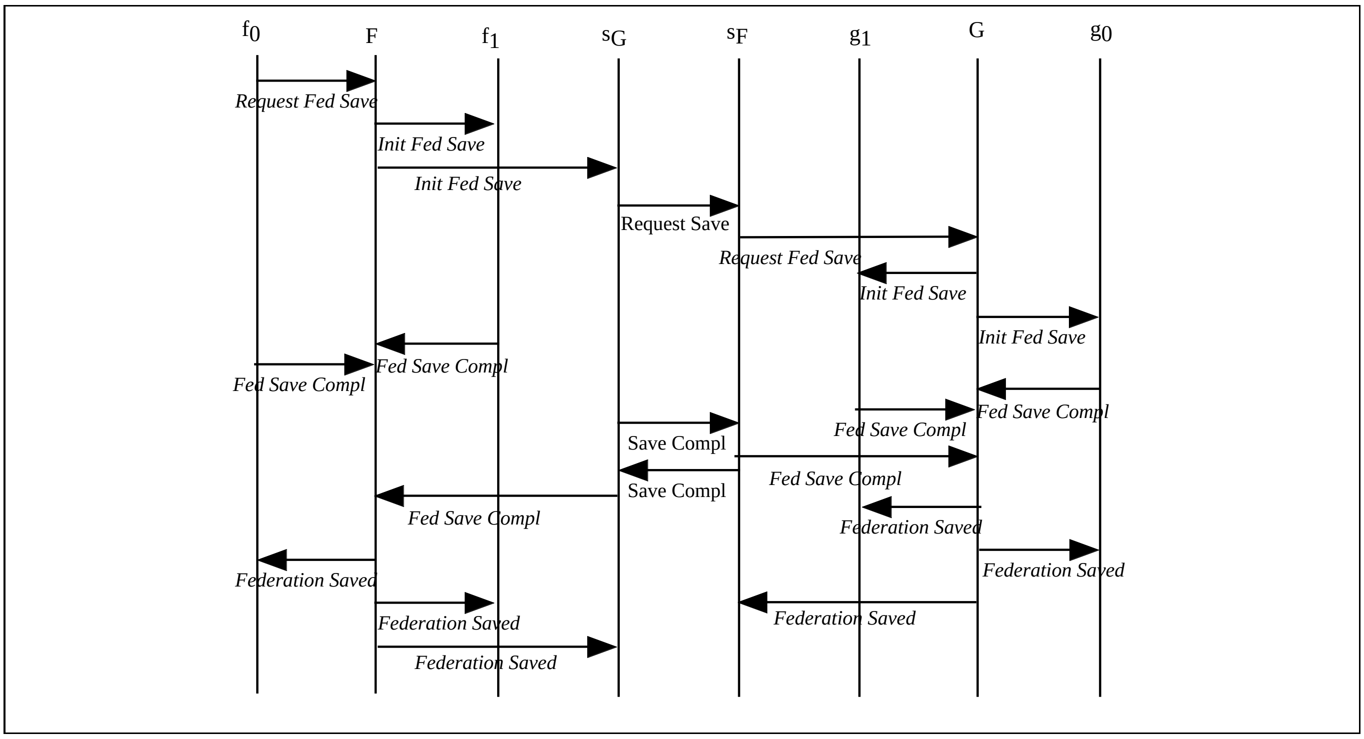

Figure 5: Example of save protocol for bridged case

synchronization notification before other federates have synchronized. In that case, it would be possible for a federate in one federation to register another synchronization before the previous one is completed in other federations. Thus, federate f may (unexpectedly) receive another synchronization request while a previous one is still pending. We refer to problems of this kind as service barrier problems.

5.2 Unexpected Failure

Another source of problems is associated with the handling failures that arise in the course of a service protocol.

For example, in the case of synchronization outlined above, the RTI may reject the registration of a synchronization point by a federate. If, for instance, the name chosen by the federate to identify the synchronization point is already used, the RTI will refuse registration.

The possibility of failure introduces yet another problem into the synchronization protocol. Consider the protocol in Figure 6. After a federate in federation F has registered a synchronization point, the surrogate s_F representing F must also register the synchronization point in federation G. The RTI for federation G may reject the synchronization point. In this situation, the surrogate s_G has no means to report the problem to the original RTI. More precisely, the HLA standard [3] is devoid of any mechanism that would allow s_G to communicate the registration failure to its RTI, since it assumes that all such failures would originate from the RTI itself.

In general, this problem is caused by the following mismatch: On the one hand, the behaviour of a surrogate is restricted to that of a federate. But on the other hand, a surrogate must also represent an entire federation adequately. If the entire federation can fail to complete an action that the HLA standard expects each individual federate to complete successfully, the standard will not offer the surrogate any mechanism to report the failure. We refer to this situation as the unexpected failure problem.

5.3 Selective Addressing

In early versions of the HLA [3], the federate registering a synchronization point optionally could indicate a set of federates. If this parameter was supplied, only those federates indicated were informed of the synchronization point and only they were required to respond to the request. This mechanism removed the need for all federates to know how to respond to every synchronization.

This kind of selective addressing leads to another class of problem, leading to conflicts with the desired properties of a bridge expressed in Section 3. According to the first property of that section, the bridge is supposed to hide the number and identities of the federates on the other side of a bridge. Since the identity of federates on the other side of a bridge are hidden, inherently a federate cannot address any of those federates individually. Indeed, the only options are for a federate to including none or all of the federates on the other side of the bridge. The first option is achieved by excluding the surrogate (of the federates on the other side of the bridge) from the synchronization set. If, on the other hand, the surrogate is included, all of the represented federates will be notified of the synchronization request and must be able to respond appropriately. Hence a bridged federation cannot have the full capabilities of a single federation.

Every HLA event that allows selective addressing exhibits the problem described above in the presence of a bridge. We refer to this problem as the selective addressing problem.

5.4 State/Behavior Problems

The HLA, like other pub-sub systems, makes assumptions about the behavior of the federates based on shared state. One instance

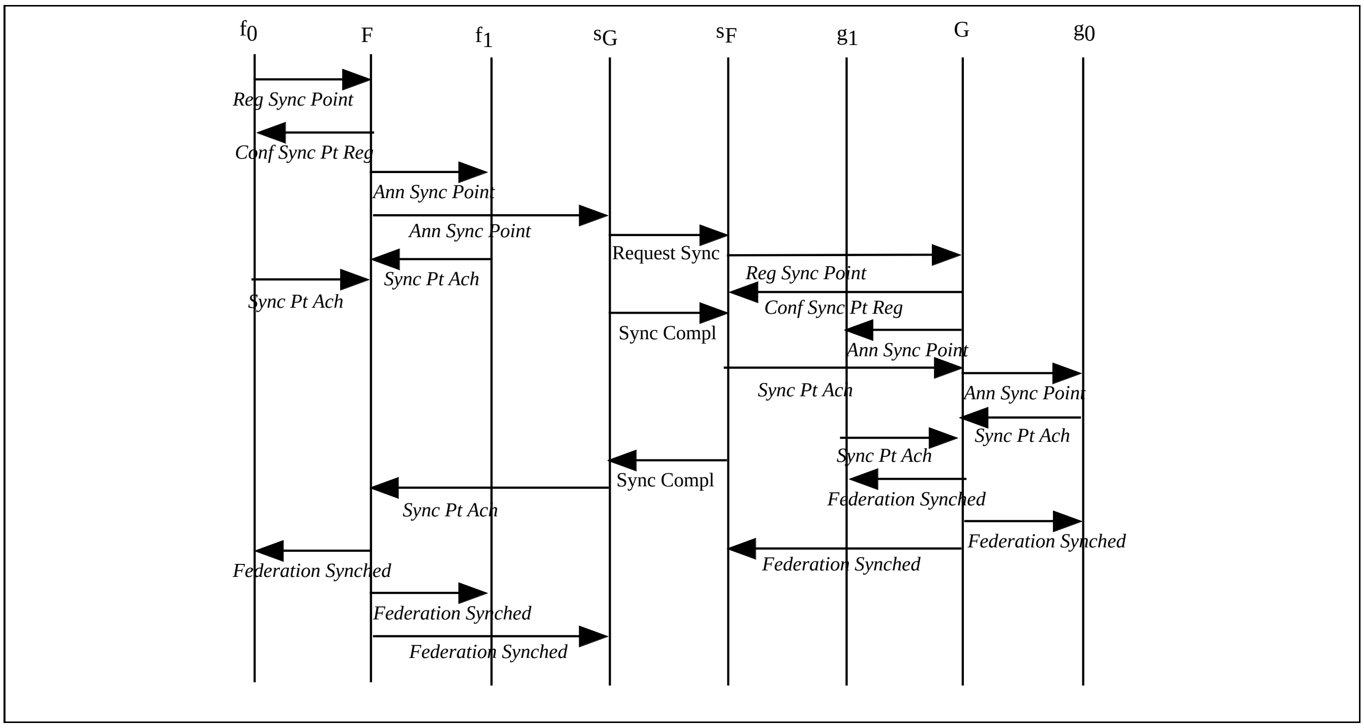

Figure 6: Example of synchronization protocol for bridged case

of shared state relates to ownership of attributes.[2] Within the HLA, a particular attribute for a particular object being modelled can be owned by a single federate, or be unowned altogether. Ownership by more than one federate is never allowed; this restriction is crucial to the proper functioning of the HLA. While any number of federates may *subscribe* to that attribute (meaning they will be notified of attribute update events), the owning federate is the only federate that may *publish* updates for that attribute.

An owning federate is free to divest ownership in that attribute at almost any time. The HLA includes two divestiture protocols: conditional and unconditional. With conditional divestiture, the federate retains ownership until another federate is willing to claim ownership. A federate executing unconditional divestiture loses ownership even if no other federate is willing to take ownership, thus introducing an unowned attribute.

In a bridged environment, a surrogate acts as the owner of all attributes modeled by any federate in the federation it represents. The bridge could choose two possible policies for handling ownership transfer: allowing or disallowing ownership to transfer across the bridge.

Either policy introduces difficulties during or after the divestiture protocol. If the bridge allows transfer, the surrogate must conditionally divest its ownership when the true owner conditionally divests. This divestiture allows a federate on the other side of the bridge to claim ownership. If two federates attempt to claim ownership, a critical race condition arises. Using standard HLA protocols, if both federates are on the same side of the bridge, the HLA protocols allow the RTI to arbitrate the winner. However, when federates are on different sides of the bridge request ownership, no single entity can arbitrate the race condition. Consequently it is possible that two federates can each believe they own the same attribute, violating a critical state invariant.

[2] In the case of the HLA, shared state is expressed as a shared simulation object model, the FOM described earlier. In other pub-sub systems shared state might be represented by things like shared files in a file system, or the contents of documents pointed to by URLs.

If the alternative policy is implemented (that is, ownership is never transferred across the bridge), the surrogate will never divest ownership of an attribute. If the true owner unconditionally divests, the attribute becomes unowned. But the surrogate still claims to be the owner and will be expected to generate new values for the attribute when so requested by the RTI. Thus, again, we have a serious inconsistency.

These problems are introduced because the actual state of the surrogate reflects an intermediate state not considered in the original protocol. As a result, the surrogate must declare itself to be in some closely-related state. For any incorrect state, the system may make assumptions about legal behaviors that conflict with the actual state of surrogate. We use the name state/behavior problems to describe situations such as these.

5.5 Unavailable Information

Although federates and surrogates serve largely different purposes, they should have the same interface: that is, their external behaviors should be identical. As we saw earlier, the unexpected failure problem violates this property. There is also a conflict when the federate interface specification cannot provide a surrogate with all the information needed to perform some bridge-specific task.

More precisely, the state of a surrogate must reflect the state of all federates in the federation being represented. In addition, the surrogate must also sometimes reflect the state of the RTI in that federation. Because no federate has a need for that information in a traditional, unbridged federation, no services are provided by the HLA to provide the surrogate access to that information. We refer to a problem as an unavailable information problem when no service is defined that can supply the information required by a surrogate.

For an instance of the unavailable information problem, suppose, for example, that a surrogate s_G of a bridge

$$(s_G, TM, s_F)$$

joins a federation F (Figure 2). The bridge needs the relevant information of F so that surrogate s_F can adequately reflect it. Relevant information includes, for instance, the current ownership of the attributes defined in the FOM. However, there is no service to supply the bridge with that information.

6. SOLUTION CATEGORIES

For problems in each of these categories, solutions, or at least partial solutions, do exist. As with the problems, we divide the solutions into general categories. For most problem categories, many, if not all, of the solution categories are meaningful. Each solution offers different advantages and disadvantages. By providing the designer with an array of approaches to solving the problems we have outlined, we hope that the designer can find a solution that fits the needs of the specific problem in the context of a specific system.

6.1 Add Services

An obvious solution to many of the problems is to add additional capabilities or services to the infrastructure.

In the case of "unavailable information" and "unexpected failure problems" it is relatively clear how additional services can help solve the problem, by making it possible for any federate to obtain more information about the state of a federation. For instance, selective addressing problems can be addressed with a new capability, a set whose exact membership is determined by the federation within which it is considered. Federates could register to belong to certain named sets or could be selected by some form of query over RTI-understood properties of them.

As indicated in Section 4.4, consensus problems can be resolved by a service that will notify a federate when it becomes the only non-consenting federate for some issue. As long as no cycles are allowed in the topology of the bridges, this service would resolve all consensus deadlock problems. In the context of the save protocol discussed in Section 4.4, for instance, the RTI could issue a message *Only One Not Yet Saved* † to inform a federate when it has become the only federate that has not yet reported a successful save. A receiving surrogate could then safely send a *Save Complete* across the bridge to the representing surrogate. In [7], the resulting protocol is shown to work for binary bridges that connect federations in a linear, non-cyclic fashion.

If the RTI cannot tell surrogates from federates, the message *Only One Not Yet Saved* † would have to be sent to every federate in the federation. If, however, the identity of the surrogates is disclosed to the RTI, only the surrogate would need to receive the message.

6.2 Smart Bridge

Adding new services and capabilities is not without cost, however. Increasing the set of messages that federates must understand is expensive, both for upgrading existing federate implementations and for future development of new federate implementations. This expense may lead to significant (and understandable) opposition from the user community to such changes in the protocols. In the context of the save protocol, for instance, federates may have to be told to ignore *Only One Not Yet Saved* † messages.

An alternative to placing the cost of supporting bridges on the federates and the RTI places the cost on the bridge. Many of these problems vanish with a sufficiently clever bridge implementation. For the unowned attribute case of the state/behavior problem category, the bridge could simply remember the last value for every attribute on every object. The surrogate can therefore provide the expected modeling behavior assumed by its ownership of the attribute.

As a second example of bridge cleverness, consider solving the service barrier problem with the bridge. If the bridge encodes the legal sequences of actions, it can buffer illegal activities until after the required consensus has been announced. To solve problems such as the consensus problem, the bridge also needs the ability to query the state of the RTI to check on the status of all other federates. Once the surrogate realizes that all other federates have achieved consensus, the bridge can indicate that the other surrogate should announce its consent.

In combination with other new, but simple, mechanisms, an arbitrarily complicated bridge can replicate the entire behavior of an RTI, allowing it to solve almost any problem that might arise in a bridged environment. One of our initial goals, however, was to build a simple bridge. If the bridge takes on the complexity of a full RTI, many of the advantages of the bridged approach vanish.

6.3 Restrict Usage

The designer of a specific pub-sub system may be able to resolve some of these problems by imposing policy on how the system uses the infrastructure provided. For example, the use of selective addressing could be limited to sets of federates within one federation.

Not all problems are amenable to this approach, however. For example, no policy will prevent the critical race potentially allowing an illegal action to take place before consensus has been achieved.

6.4 Ignore Problem

The final approach to solving these problems is the simplest to implement: simply ignore the problem. For some service barrier problems, the risk of the critical race being won by the illegal action may be negligible. For other service barrier problems, the outcome of an "illegal" action may not be a severe problem. For unexpected failure problems, the chance of failure or significance of the failure may not be worth the cost of extending the model to support the failure. For a state/behavior problem such as the unowned attribute problem addressed earlier, the cost of never responding to a request may be minimal.

Of course, ignoring the problem should not be done lightly. For some problems, ignoring the problem could have disastrous effects. For example, if the result introduces two owners of a given attribute or a deadlock in seeking consensus, ignoring the problem could invalidate the on-going simulation.

Although no category offers an ideal solution for every problem, the combination of solution categories offers a substantial arsenal for the designer. By appropriately choosing solutions from these categories for each problem encountered, a designer can make effective use of a bridge to join multiple pub-sub federations.

7. CONCLUSION AND FUTURE WORK

In this paper, we have described six problem classes associated with the design of a lightweight bridge composition for distributed pub-sub federations. We have also described four possible approaches to dealing with these problems, providing one or more possible solutions for an example problem from each problem category.

Although grounded in the details of a widely-used, standardized pub-sub framework, we claim that these problems are applicable to the more general problem of composing pub-sub systems. We believe this because most pub-sub systems do much more than dispatch and filter events. To have a viable pub-sub system, one must also worry about other forms of component synchronization and coordination. Sometimes these capabilities are provided directly by the run-time infrastructure (as in the case of the HLA RTI). In other cases, it is layered as protocols and conventions on top of a basic event transport mechanism. But in either case, problems of protocol extensibility for multiple pub-sub federations will become an important problem. Indeed, we suspect that such problems will be true of any large scale system of distributed asynchronous components.

In our work to date, we have not focused on timing issues that may arise from the introduction of bridges. We have restricted our consideration to new critical race issues that may invalidate the protocols. Beyond the obvious performance issues, the timing changes introduced by bridges may change the order that federates observe events and thereby introduce otherwise unknown semantic conditions. While these conditions will be legal in an absolute sense, they may expose flaws in the federates that would not be exposed in an unbridged system. Additional investigations in the timing issues may uncover other types of problems.

Of course, using a bridge is not the only way of combining multiple pub-sub federations. One might, for example join the RTIs directly using some form of RTI-to-RTI communication link. That approach has the advantage that RTI state could be communicated directly without going through the component (federate) interface. But it also has the disadvantage that it requires considerable cooperation at the RTI implementation level. This, in turn, would make it hard to integrate federations that use RTI implementations created by different vendors. It also would make it harder to localize policies of filtering and transformation. Nonetheless, investigating such lower-level bridging mechanisms would be an interesting direction for future work.

Other avenues of future work include the use of formal models to demonstrate existence of problems, and to verify the correctness of the solutions that we have proposed. Looking for other problem classes might also be possible, using, for example, the exhaustive search capabilities of model checkers.

In this paper, we focused on the case in which bridges connect federations in a linear, acyclic fashion. Other topologies would also be worth exploring. We have done some work on investigating problems that arise when one allows cyclic topologies and identified some problems that don't arise in the simple case [6]. A more complete characterization of topologies and problem classes would be a valuable extension of our work.

Finally, investigating the use of bridges for other architectural styles would be worth pursuing. For example, one might investigate lightweight compositional mechanisms for architectures based on asynchronous (point-to-point) messages, or shared data approaches.

8. REFERENCES

[1] R. Allen, D. Garlan, and J. Ivers. Formal modeling and analysis of the HLA component integration standard. In *Sixth International Symposium on the Foundation of Software Engineering (FSE 6)*, pages 209–221, Lake Buena Vista, Florida, November 1998. ACM Press.

[2] A. Carzaniga, D. Rosenblum, and A. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 219–227, Portland, OR, July 2000.

[3] S. I. S. Committee. *IEEE P1516.1/D4 Draft Standard for Modeling and Simulation, High Level Architecture — Federate Interface Specification*. IEEE Computer Society, 1999.

[4] The Common Object Request Broker: Architecture and specification. OMG Document Number 91.12.1, December 1991. Revision 1.1 (Draft 10).

[5] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Designs (Third Edition)*. Addison Wesley, Pearson Education, 2001.

[6] C. Damon, R. Melton, R. Allen, E. Bigelow, J. Ivers, and D. Garlan. Formalizing a specification for analysis: The HLA ownership properties. Technical Report CMU-CS-98-149, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1998.

[7] J. Dingel, D. Garlan, and C. Damon. A feasibility study of the hla bridge. Technical Report CMU-CS-01-103, Department of Computer Science, Carnegie Mellon University, March 2001.

[8] J. Dingel, D. Garlan, S. Jha, and D. Notkin. Reasoning about implicit invocation. In *Sixth International Symposium on the Foundation of Software Engineering (FSE 6)*, pages 209–221, Lake Buena Vista, Florida, November 1998. ACM Press.

[9] J. Dingel, D. Garlan, S. Jha, and D. Notkin. Towards a formal treatment of implicit invocation using rely/guarantee reasoning. *Formal Aspects of Computing*, 10:193–213, 1998.

[10] W. Edwards and T. Rodden. *Jini Example by Example*. Sun Microsystems Press, 2001.

[11] IEEE. *Proceedings of Winter Simulation Conferences (WSC'95 - WSC'01)*. ACM Press, 1995-2001.

[12] H. Jubin. *JavaBeans By Example*. Upper Saddle River: Prentice Hall, 1998.

[13] F. Kuhl, R. Weatherly, J. Dahmann, and A. Jones. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Prentice Hall, October 1999.

[14] S. Mullender. *Distributed Systems (Second Edition)*. Addison Wesley, 1993.

[15] S. Reiss. Connecting tools using message passing in the FIELD program development environment. *IEEE Software*, July 1990.

[16] R. Taylor, N. Medvidovic, K. Anderson, E. W. Jr., J. Robbins, K. Nies, P. Oreizy, and D. Dubrow. A component- and message-based architectural style for gui software. *IEEE Transactions on Software Engineering*, June 1996.