HLA: A Standards Effort as Architectural Style

Robert Allen
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
rallen@cs.cmu.edu

Abstract

In this position paper we introduce a case study, the DoD "High Level Architecture for Simulations (HLA)," and briefly discuss our efforts to apply WRIGHT, an architectural description language, to the HLA. Our work on HLA has focused on understanding the HLA as an architectural style, concentrating on the Interface Specification (IFSpec) description of the "Runtime Infrastructure (RTI)" as the central architectural design issue. Specifically, we have used WRIGHT, a formal architectural description language, to characterize the RTI and analyze a number of its properties. By providing an analysis of the properties of the RTI as described by the IFSpec, we can help the standards committee to determine whether the IFSpec ensures the properties that they want and to discover inconsistencies or other weaknesses of the specification.

1 Introduction

An important challenge to software architecture researchers is to provide practical tools and analyses that can have an impact on important, large-scale software development efforts. Unless the value of architectural approaches can be shown on significant problems, rigourous software architecture will remain merely an academic exercise.

In this position paper we introduce a case study, the DoD "High Level Architecture for Simulations (HLA)," and briefly discuss our efforts to apply WRIGHT [1, 2], an architectural description language, to the HLA. Our work on HLA has focused on understanding the HLA as an architectural style, concentrating on the Interface Specification (IFSpec) description of the "Runtime Infrastructure (RTI)" as the central architectural design issue. Specifically, we have used WRIGHT, a formal architectural description language, to characterize the RTI and analyze a number of its properties. By providing an analysis of the properties of the RTI as described by the IFSpec, we can help the standards committee to determine whether the IFSpec ensures the properties that they want and to discover inconsistencies or other weaknesses of the specification.

Appears in: Proceedings of the Second International Software Architecture Workshop (ISAW-2), San Fransisco, CA, 14-15 October, 1996.

2 Motivation and Overview of HLA

Simulation is an important tool for the military, both in preparing for and in carrying out its various missions throughout the world. Simulations are used in personnel training, design and testing of equipment, and analysis of both past and future actions. The U.S. DoD has made a considerable investment in equipment and software for simulation – one effort alone, directed at army training, cost $2 billion [3].

Given the number and complexity of these simulations, developers are faced with a daunting task: to provide software that meets the challenges posed by the high-fidelity, real-time, physically distributed, mission-critical simulation domain, and yet to minimize redundancy of effort across applications and maximize flexibility of the software to be used for new, possibly unanticipated tasks.

An approach that has been applied with success in similar situations is to define a domain-specific software architectural style [4]. An architectural style guides the development of a family of software systems by providing a common architectural vocabulary that can be used to describe the structure of individual systems and constraining the use of that vocabulary. Use of a style helps the architect make guarantees that are critical in the particular domain and analyze a system for properties considered of especial importance in that style.

A typical architectural style provides both a vocabulary for the description of architectural components, the separate computations that are combined to form a complete system, and a system of architectural connectors, patterns of interaction among components. A style is often supported by infrastructure that eases the implementation of components and provides support for their interaction via the style's connectors.

With these potential benefits in mind, the Defense Modelling and Simulation Office (DMSO) has undertaken the development of the "High Level Architecture for Simulations" (HLA) [5]. The HLA is intended to support the coordination of different simulations with the goal of simplifying the integration task, encouraging modularization of simulations, and increasing quality of simulations and potential for reuse.

The HLA defines a standard for the coordination of individual simulations through the communication of data object attributes and events. In HLA, members of a *federation* — the HLA term for a distributed simulation — coordinate their models of parts of the world through sharing objects of interest and the attributes that define them. Each member of the federation (termed a *federate*) is responsible for calculating some part of the larger simulation and

broadcasts updates using the facilities of the Runtime Infrastructure (RTI). Messages both from the federates, *e.g.* to indicate new data values, and to the federates, *e.g.* to request updates for a particular attribute, are defined in the "Interface Specification" document. Each message is defined by a name, a set of parameters, a possible return value, pre and post conditions, and a set of exceptions that may occur during execution of the message.

The interface is divided into five parts: Federation management, declaration management, object management, ownership management, and time management. The federation messages are used by federates to initiate a federation execution, to join or leave an execution in progress, to pause and resume, and to handle saves of execution state. Declaration messages are used to communicate about what kinds of object attributes are available and of interest, while object messages communicate actual object values. Ownership messages are used in situations when one federate has been responsible for calculating the value of an object attribute but for some reason another federate should now take over that responsibility. (Example situations include when the original federate must drop out or when some property of the object indicates that the new federate is better able to support that object. For example, if a unit moves from one geographic region to another, then simulators responsible for modelling troops in each region might hand off ownership of the unit's representation object.) The fifth category, time management, is used to keep each member of the federation synchronized, either by maintaining correspondence of wall-clock time, by lock-step advancement of a logical time, or by other means.

The intention of the interface specification is that the general standard be refined into multiple implementations depending on the various needs of particular simulation domains. For example, different simulations would have different performance constraints, requirements for physical distribution, and models of time-synchronization, depending on the scale and use of the simulation. In addition, each federation needs to augment the standard with its own detailed object-model to ensure semantically consistent exchange of data between federates.

For example, as part of the current standard development effort, several implementation efforts, each termed a *proto-federation*, are underway. One proto-federation effort is described in [6].

3 A WRIGHT Analysis of the RTI

WRIGHT is a formal language for describing software architecture. There are two main kinds of description in software architecture — architectural style and architectural configurations. An architectural configuration is just what you would expect: a description of the structure of a single software system, in terms of decomposition into encapsulated computations (components) and interaction pathways among the elements (connectors). An architectural style describes the structure and common properties of a family of systems rather than of an individual system. A style provides a vocabulary for describing components and a set of (possibly parameterized) connectors that can be used to compose the components into configurations (which are instances of the architectural style). For example, the pipe-filter style describes the vocabulary of sequential data input and output used in filter components and the pipe interaction mechanism. A configuration in the pipe-filter style would provide a specific set of filters that appear in the particular system being described and bind the outputs of specific filters to the inputs of other filters (via pipes). Thus, the configuration could be represented as a "box and line" diagram, where the boxes are filters and the lines are instances of the pipe interaction pattern (or connector).

In addition to the introduction of vocabulary, an architectural style may constrain how configurations are assembled. For example, a "pipeline" style, a specialization of the pipe-filter style, would specify that there must be a single sequence of filters, where each filter sends its output to the next filter in line.

While WRIGHT is capable of describing both styles and configurations, and of relating the two (confirming that a particular configuration is, or is not, an instance of a style), the HLA formalization has focussed on specifying the IFSpec as a style. That makes sense because it is a guideline for clarifying the construction and behavior of many different federations. Each federation would be a configuration in the "HLA style". (Or rather, the parts of a federation that are selected for a particular federation execution would be such a configuration.)

The basic elements of the HLA formalization consist of the introduction of a single component type, the *federate*, a single connector, the *RTI*, and a configuration constraint rule, that there shall be a single RTI connector and all federates shall interact using it.

While the overall specification is considerably larger than can be shown in a short position paper, a few extracts will give the flavor. In order to specify the properties that are required of any federate to participate in an HLA simulation, an *interface type* is introduced, the *SimInterface*, that defines what the communication behavior of the federate will be. The SimInterface introduces the various messages that will pass between the federate and the RTI. Messages (represented by *events* in WRIGHT) are divided into those that are initiated by a federate, such as $\overline{\mathsf{joinFedExecution}}$, which indicates that the federation wishes to participate in the simulation, and those that are initiated by the RTI, such as reflectAttributeValues, which is used to inform a federate of new data values. The presence of an overbar (as in $\overline{\mathsf{e}}$) indicates an event that is *initiated* by the process. An undecorated event (as in e) indicates an observation of the activity of some other process. The various events are combined using process-algebra operators (as in CSP [7]) to indicate any constraints on ordering of events. An extract of the *SimInterface* definition is as follows:

$$
\begin{array}{l}
\textbf{Interface Type}\ \mathsf{SimInterface} = \mathsf{JoinFed} \\
\qquad\qquad\qquad\qquad \sqcap\ \overline{\mathsf{createFedExecution}} \rightarrow \mathsf{JoinFed} \\
\quad \textbf{where} \\
\qquad \mathsf{JoinFed} = \overline{\mathsf{joinFedExecution}} \rightarrow \mathsf{NormalExecution} \\
\qquad \mathsf{NormalExecution} = \mathsf{reflectAttributeValues} \rightarrow \mathsf{NormalExecution} \\
\qquad\qquad\qquad\qquad\quad \square \ldots \\
\qquad\qquad\qquad\qquad\quad \sqcap\ \overline{\mathsf{resignFedExecution}} \rightarrow \S
\end{array}
$$

This extract indicates that before joining an execution, the federate may need to create it (if no other federation has), and that it must indicate the start of computation by an explicit joinFedExecution message. The federate is then in the condition NormalExecution where it can both send and receive messages from the RTI. Finally, the federation may, during normal execution, choose to resign from the execution (indicated by the resignFedExecution message), after which it must not send or receive any more messages.

While the SimInterface models the behavior of a single federate, the RTI describes how multiple federates interact. In the connector specification, the **Glue** provides a specification indicating how events of one component relate to those of the others. In the extract in figure 1, event names are prefixed with Fed_i to indicate that it is an event of the *ith* federate.

This extract of the RTI connector specification clarifies the specification in SimInterface that each federate has the option of creating the RTI execution: Exactly one of them must do so, and none of the others are permitted to do so. Similarly, the RTI execution must not be destroyed unless there are no joined federates, and once the RTI is destroyed, no further interaction may take place.

Connector RTI (nsims : 1..)
Role $\text{Fed}_{1..nsims}$ = SimInterface
Glue = $\forall i : 1..nsims \,\Box\, \text{Fed}_i.\text{createFedExecution} \rightarrow \text{WaitForSim}_{\{\}}$
where $\text{WaitForSim}_{\{\}} = \forall i : 1..nsims \,\Box\, \text{Fed}_i.\text{joinFedExecution} \rightarrow \text{WaitForSim}_{\{i\}}$
$\Box\, \forall i : 1..nsims \,\Box\, \text{Fed}_i.\text{destroyFedExecution} \rightarrow \S$
$\text{WaitForSim}_{ActiveFeds} = \forall i : 1..nsims \,\Box\, \text{Fed}_i.\text{joinFedExecution} \rightarrow \text{WaitForSim}_{ActiveFeds \cup \{i\}}$
$\Box\, \forall i : ActiveFeds \,\Box\, \text{Fed}_i.\text{updateAttributeValues}$
$\rightarrow \langle send\ reflectAttrValues.. \rangle$
$\rightarrow \text{WaitForSim}_{ActiveFeds}$
$\Box\, \forall i : ActiveFeds \,\Box\, \text{Fed}_i.\text{resignFedExecution}$
$\rightarrow \text{WaitForSim}_{ActiveFeds \setminus \{i\}}$
$\Box$ ...

Figure 1: An extract of the RTI Connector.

4 Impacts of Formalization Effort

Because we have been formalizing the HLA as an architectural style, analysis of our WRIGHT specification informs us about properties of the IFSpec in general, not of any particular proto-federation. That is, if we discover a property of the specification and prove that it holds, it must hold for every federation that obeys the IFSpec. If one of the proto-federation efforts discovers a problem in their implementation, there is no intrinsic indication of whether it is fundamental to the IFSpec, permitted by the IFSpec but not necessarily true of every implementation, or an indication that the prototype implementation is in violation of the IFSpec. With the WRIGHT specification, the analysis will indicate whether the property is intrinsic or only a possibility. Because the specification is formal, it is possible to verify that the specification does indeed obey the IFSpec (*e.g.* that the preconditions of a message are satisfied whenever that message is sent).

The main impact of our formalization effort is on the IFSpec itself. By providing an analysis of the properties of the IFSpec, we can help the standards committee to determine whether the IFSpec ensures the properties that they want and to discover any inconsistencies or other weaknesses of the specification.

The impact of the formalization on the IFSpec can occur in two places. First, it might help to suggest places where the RTI standard needs to be changed or strengthened, and second, it can help to provide a basis for supplemental documentation or indicate where the documentation might be elaborated even when the standard itself does not need to be changed.

As an example of the latter, consider "exceptions." Part of each message definition in the IFSpec is a list of exceptions. For example, joinFedExecution includes the exception "federate already joined." As part of our preliminary attempts to formalize the HLA, we realized that the formalization (and presumably any implementation) wasn't possible unless we knew if these exceptions resulted in actual message traffic or whether they were simply anomalies that should be considered (but without explicit notification). As a result of this observation, the new IFSpec draft clarifies this point in the glossary, resolving the ambiguity, which could have led to unnecessary conflicts between federation implementations.

Once we have determined enough detail about the specification to formalize it, WRIGHT can be used to detect potential conflicts from, for example:

- insufficient preconditions
- missing information
- race conditions
- wrong directionality (parameter v. return value, RTI v. federate initiation)

As an example of missing information, consider the example of the destroyFedExecution message, which indicates that the current execution is done and that the RTI should terminate. This message has a precondition that ensures that this message is safe to execute: there must be no joined federates. But there is no way, using the IFSpec as written, for a federate to determine what the set of joined federates is. Thus, there must be some external way for a federate to get this information before sending the message. (For example, a federation might define a special "query" message, the federate might be a person who also controls starting and stopping all of the simulation federates, or the federate might simply ignore the precondition and count on getting an exception whenever there is anything still executing.)

One way that this example analysis might impact the IFSpec effort is to suggest a core set of query messages that all federations must supply. If that solution isn't satisfactory to everyone (which wouldn't be surprising), then some kind of documentation (a federation developer's guide?) might want to be considered that would discuss these kinds of issues ("the following ambiguities in the IFSpec are deliberate and must be resolved by any federation... Possible solutions include...") Obviously the formalization can't resolve the pragmatics of various options, but it does provide a means of exposing possible starting points.

5 Conclusion

This paper has briefly introduced the DoD "High Level Architecture for Simulations," a draft standard for the integration of distributed simulations, and described one effort to formalize it as an architectural style using the architecture description language WRIGHT.

The HLA represents a potentially important tool for the developers of simulations, and in order to meet its goals, many important issues, not limited to those discussed above, must be addressed. These issues include: the consistency and precision of the interface definition; how to determine compliance with the standard while permitting necessary variation both in the semantic basis of the simulation, meeting performance constraints, and differing models of time; refinement of the standard into an implementation infrastructure; methods of analysis of federations; effect of the standard on individual federates, including pre-existing (legacy) simulations.

These are architectural issues — the decisions made at this level about the HLA and the constraints placed on individual implementations will control the structure of every simulation constructed

using the standard, and thus form the basis for their success or failure. It is up to us as software architecture researchers to take up the challenge and provide sound, practical tools and techniques to support efforts such as this one.

References

[1] Robert Allen and David Garlan. The WRIGHT architectural description language. Technical report, Carnegie Mellon University School of Computer Science. in preparation.

[2] Robert Allen and David Garlan. Formalizing architectural connection. In *Proceedings of the Sixteenth International Conference on Software Engineering*, May 1994.

[3] Peter Brooks. New directions in advanced distributed simulation, April 1996. Presentation at CMU.

[4] *Proceedings of the Workshop on Domain-Specific Software Architectures*, Hidden Vallen, PA, July 1990. Software Engineering Institute.

[5] DMSO. Web site, URL = http://www.dmso.mil/docslib/hla/.

[6] Peter Green and Terry Griffin. Specification for the rtis hla/rti implementation. Technical Report RTIS10951, The Real-Time Intelligent Systems Corporation, Westborough, MA, October 1995.

[7] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.