Formalizing a Specification for Analysis: The HLA Ownership Properties

Craig A. Damon, Ralph Melton, Robert J. Allen,
Elizabeth Bigelow, James M. Ivers, David Garlan

April, 1999
CMU-CS-99-126

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This research was supported by the Defense Advanced Research Projects Agency and Rome Laboratory, USAF, under Cooperative Agreement F30602-97-2-0031, and by the Defense Modeling and Simulation Office (DMSO). Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Rome Laboratory, the US Department of Defense, or DMSO. The US Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation thereon.

We would like to acknowledge the help of Richard Weatherly and Reed Little, our main sources of wisdom for the intended behavior of the HLA.

Keywords

Formal specification, model checking, Z specification language, distributed simulation.

Abstract

Interfaces are commonly specified using informal or semi-formal techniques, relying primarily on natural language descriptions. Such specifications, however, can easily overlook significant details and are not amenable to analysis by automated tools. This paper looks at formalizing one portion of a substantial specification, the ownership management chapter of the DoD HLA framework, and at the subsequent analysis using the tool Ladybug.

1. Introduction

Developers typically specify interfaces informally, or perhaps semi-formally, relying primarily on a natural language description of each function in the interface to explain its operation and interactions. These informal specifications are undeniably simpler and faster to produce than ones making significant usage of a formal notation, such as Z or CSP. The informal specifications are also accessible to a much broader audience; relatively few people, even among well-trained practicing software engineers, are fluent in any of the formal notations.

There are, however, advantages gained from the effort of producing a formal specification. A clarity of precision is the most obvious of these advantages. Whereas a natural language description can often be ambiguous, a formal notation definition is precise and unambiguous. The rigor of formalization may also force the consideration of poorly understood areas that are easily swept under the rug of natural discourse.

A second advantage of formalization is the opportunity to use automated analysis tools to check certain desired properties of the design. Model checkers and similar automated tools now exist for many formal notations [CW+96]. These tools accept a formal description of a system and one or more claims about that system as their input, outputting a concrete counter-example for each claim that is found to be false.

This paper describes the formalization into Z of one portion of the semi-formal specification of a large system, the HLA distributed simulation architecture. The goal of this formalization was to find inconsistencies and ambiguities in the original specification, through the formalization itself as well as through the use of an automated analysis tool, Ladybug. This paper documents five of the issues discovered in this process, any of which could have led to incompatibilities in apparently conforming components.

1.1 Overview of HLA

Beginning in 1996, the Defense Modeling and Simulation Office (DMSO) of the United States Department of Defense developed a component integration standard for distributed simulation called the "High Level Architecture" (HLA). Informally, the HLA prescribes a kind of "simulation bus" into which simulations can be "plugged" to produce a joint (distributed) simulation. A goal of the standard was to allow independent vendors to develop simulations that can be combined for use in a single, unified simulation with minimal complications.

In the HLA design, members of a *federation* — the HLA term for a distributed simulation — coordinate their models of parts of the world by sharing objects of interest and the attributes that define them. Each member of the federation is called a *federate*. A federate is responsible for calculating some part of the larger simulation and broadcasting updates using the facilities of the runtime infrastructure, termed the *RTI*.

Routines that support communication both from the federates (e.g., to indicate new values) and to the federates (e.g., to request updates for a particular attribute) are defined in the "Interface Specification" document or *IFSpec* [DoD97]. Routines, or "services", in the IFSpec are defined by a name, the initiator (either a federate or the RTI), a set of parameters, a possible return value, pre- and post-conditions, and a list of exceptions that may occur as a result of executing the service.

An example of a typical RTI service is shown in Figure 1 (taken from [DoD97]). This service is initiated by a federate when it wants to relinquish ownership values for some attributes of a particular object being simulated by the federate. The federate relinquishes ownership, however, only when informed by the RTI using the *Attribute Divestiture Notification* service.

The HLA is a complex integration framework. The current IFSpec includes over 125 different services, and the full document is over 400 pages of description. While the part of the HLA design that deals with attribute broadcast is relatively straightforward, the overall framework is complicated significantly by the need to deal with issues such as starting, stopping, and pausing; allowing one federate to transfer object ownership to another; and distributed clock management and time-ordered message sequencing.

To make the integration framework manageable, the IFSpec is divided into six chapters: federation management, declaration management, object management, ownership management, time management, and data distribution management. Federates use the federation management services to initiate a federation execution, to join or leave an execution in progress, to pause and resume, and to save execution state. Declaration management services communicate what kinds of object attributes are available and of interest, whereas object management services communicate actual object values. Ownership management services allow responsibility for calculating the value of an attribute to be transferred from one federate to another. Time management services coordinate the logical time advancements of federates and to ensure that messages are delivered in time-stamp order. Data distribution management services filter attribute updates for each federate based on defined criteria, reducing message traffic and processing requirements.

1.2 The HLA Model of Attribute Ownership

The remainder of this paper focuses on the ownership management services, which control the transfer of ownership of attributes. The HLA adopts an object view of a distributed simulation; the simulation universe consists of a collection of objects, each of which has a set of attributes. The job of the overall simulation is to calculate and update values of these attributes over time. Different federates can calculate values of different attributes of the same underlying object.

Every object in an HLA simulation is an instance of some object class[1]. These classes define the attributes for their instances; the IFSpec defines an attribute to be “a distinct, identifiable portion of the object state”. Version 1.2 of the IFSpec is at times inconsistent in its usage of the term attribute, sometimes meaning a value associated with a single object and at other times meaning the generic attribute across all objects of a particular class. We will use the phrase *object attribute* to describe the former case and *class attribute* to describe the latter case.[2]

The object classes defined by the IFSpec support inheritance, which introduces some additional complexity. None of the properties we model depend in any way upon this inheritance, so we

1. The IFSpec discusses two kinds of classes: object classes and interaction classes. Interaction classes are irrelevant to the work presented here, so we will use the word class to refer to object classes.
2. Partially in response to our concerns about this distinction, later versions of the IFSpec consistently distinguish object attributes from class attributes. They chose, however, to use the term instance attribute rather than object attribute.

5.1 Request Attribute Ownership Divestiture

Federate Initiated

Notifies the RTI that the federate no longer wants to own the specified attributes of the specified object. The federate supplies an object ID and set of attribute designators.

Options:

1. The federate can specify which federate(s) can take ownership of the released attributes, otherwise any federate may own them.
2. The federate can indicate if the requested ownership divestiture is to be negotiated or unconditional. If the divestiture is negotiated, ownership will be transferred only if some federate(s) accepts. An unconditional transfer will relieve the divesting federate of the ownership, causing the attribute(s) to go into (possibly temporarily) the unowned state, without regard to the existence of an accepting federate.

The federate must continue its publication responsibility for the specified attributes until it receives permission to stop via the *Attribute Ownership Divestiture Notification* service. The federate may receive one or more *Attribute Ownership Divestiture Notification* invocations for each invocation of this service.

Supplied Parameters

An object ID designator

A set of attribute designators

Ownership divestiture condition (negotiated or unconditional)

A user-supplied tag

Optional set of federates

Returned Parameters

None

Pre-conditions

The federation execution exists

The federate is joined to that federation execution

An object instance with the specified ID exists

The federate owns the specified attributes

Post-conditions

No change in attribute ownership

The federate has informed the RTI of its request to divest ownership of the specified attributes

Exceptions

Object not known

Attributes not defined in the FED

Federate does not own attribute

Invalid divestiture condition

Invalid candidate federate

Federate is not a federation execution member

Save in progress

Restore in progress

RTI internal error

Related Services

Request Attribute Ownership Assumption

Attribute Ownership Divestiture Notification

Attribute Ownership Acquistion Notification

Figure 1: The Request Attribute Ownership Divestiture service of the RTI, as specified in the IFSpec [DoD97] (version 1.2).

Updates A federate updates an object attribute when it sends out a new value for the attribute. An update is an event, not a state.

Reflects A federate reflects an object attribute when it receives a new value for the attribute. A reflection is also an event, not a state.

Owns A federate owns an object attribute if it has the privilege to update values for that attribute. An attribute should have no more than one owner at a time. Ownership is a state.

Publishes A federate publishes a class attribute if it could provide updates for that kind of attribute, whether or not it currently has the privilege to do so for any particular object. Publishing is a state, not a point event. Multiple federates may publish the same class attribute at the same time.

Subscribes A federate subscribes to a class attribute if it wants to receive updates to that kind of attribute. Subscribing is a state.

Figure 2: Brief glossary of IFSpec terms

dropped consideration of inheritance. The choice of which portions of the model to consider and which to ignore is an important aspect of modeling a system for analysis.

HLA propagates object attribute updates using a publish and subscribe system, although they use the standard terminology in a somewhat non-standard way. A brief glossary of the terminology (as used in the IFSpec) is presented in Figure 2.

As an overview, an object attribute of a given object is updated only if some federate

- *publishes* the corresponding class attribute,
- *owns* the object attribute for that object, and
- *updates* the value

Another federates “sees” the updated value for the object attribute only if it *subscribes* to the corresponding class attribute. The receipt of this new value is known as a *reflection*.

A federate may own an object attribute only if it publishes the corresponding class attribute and no other federate owns that object attribute. A federate begins the acquisition of ownership of an object attribute by requesting ownership from the RTI using the *Request Attribute Ownership Acquisition* service. The RTI will respond, if possible, by granting ownership using the *Attribute Ownership Acquisition Notification* service.

A federate can similarly disown an object attribute by initiating the *Request Attribute Ownership Divestiture* service and waiting for the corresponding *Attribute Ownership Divestiture Notification* service invocation from the RTI. The divestiture request can either be unconditional, leading to a possibly unowned object attribute (which will therefore not be updated until another federate claims ownership), or it can be negotiated, with the federate maintaining ownership until the RTI can locate another federate willing to own the object attribute.

The RTI searches for possible owners of an object attribute that is being divested by invoking the *Request Attribute Ownership Assumption* service on federates that are currently publishing the corresponding class attributes. An interested federate may then be granted ownership of the object attribute by the RTI.

1.3 An Outline of the Approach Used

We described the ownership management sections of the HLA specification using Z [Spi92], a formal notation built from standard sets, functions and relations. We chose Z to model these aspects because it is particularly good at describing structural properties of a system. Other notations, such as CSP [Hoa85], are weaker in their support of structural properties, but offer strong support for describing dynamic properties of the system.

We then translated the Z notation to NP [JD96a], the input language used by the Ladybug checker. NP is essentially a first-order subset of Z with the many special characters used in Z remapped to ASCII equivalents. Ladybug[3] is a new tool that analyzes claims about a specification, producing concrete counter-examples to contradict each invalidated claim. Ladybug considers all possible cases using a small number of elements, such as objects, federates and object attributes for the HLA, as specified by the user. The analysis is sound within those user-specified bounds, meaning that Ladybug will produce a counter-example if any exist. Ladybug's predecessor, Nitpick [JD96b], was the first tool able to perform fully automatic semantic analyses of specifications written using a Z-like notation.

Special care must be taken when formalizing a specification for analysis. The simplest translation may prevent the discovery of some interesting flaws. To be checkable, the specification must clearly delineate those properties that are guaranteed to be true from those properties that are desired to be true.

Consider the ownership relationship from HLA as an example. We can model this relationship in Z as a relation mapping federates to object attributes, with each federate-attribute pair describing the ownership of a single object attribute by a federate. However, only a single federate is allowed to own a given object attribute at a time. We can encode this constraint in the Z description by making the relation injective. However, placing this encoding on the basic description of the system prevents Ladybug from discovering possible corruptions of the system involving multiple simultaneous owners. Instead, we encode this constraint as a separate property, allowing it to be checked. (See the description of *NoTwoOwners* in the next section for more details.)

Checking that the *Attribute Ownership Divestiture Notification* service maintains this property requires checking the claim

$$\mathtt{NoTwoOwners} \wedge \mathtt{AttrOwnDivestNotify} \Rightarrow \mathtt{NoTwoOwners}'$$

This formula says that if `NoTwoOwners` holds initially and `AttrOwnDivestNotify` is executed, then `NoTwoOwners` will still hold afterwards. If `NoTwoOwners` is not invariant across `AttrOwnDivestNotify`, Ladybug will provide concrete counterexamples demonstrating the violation of `NoTwoOwners`.

1.4 Related Work

This effort is far from being unique. Allen and others have formalized and analyzed other segments of the HLA system using Wright, a tool based on CSP [AGI98]. Just as the work described in this paper builds on the strengths of Z to analyze selected structural properties of the

3. Ladybug, and its predecessor Nitpick, are freely available at http://www.cs.cmu.edu/~nitpick.

$[CLASS, CLASSATTR]$

$$
\begin{array}{|l}
AttributesToClass : CLASSATTR \rightarrow CLASS \\
privToDeleteObject : CLASS \rightarrowtail CLASSATTR \\
\hline
privToDeleteObject^{\sim} \subseteq AttributesToClass
\end{array}
$$

Figure 3: Z model of classes and class attributes

HLA system, that work builds on the inherent strengths of CSP to analyze selected dynamic properties of the HLA system.

At least two other efforts have considered the formalization and analysis of systems using NP and a Ladybug-like tool. In [JD96b], Jackson formalizes the paragraph style mechanism in a word processing program, finding anomalous behaviors with the use of Nitpick. Ng also uses Nitpick to analyze a formalization of the mobile IPv6 protocol, discovering a flaw that can lead to cycles in the packet forwarding algorithm [JNW98].

2. The Formal Model

This section describes the formal model of HLA that we derived from version 1.2 of the IFSpec. To reduce the reader's burden, this section details only a representative sampling of properties and operations. The remaining properties and operations are similar to those described here. The complete Z model of the ownership management services is given in Appendix A.

The Z model of ownership management can be broken down into four major pieces:

- classes, objects, and attributes, which are global to all of HLA
- the state required (explicitly or implicitly) by the ownership management specification
- properties about the state
- operations on the state

We assume that the reader is familiar with Z notation. For readers unfamiliar with the specifics of Z, a summary of the operators used in this specification is given in Appendix B.

2.1 Classes, Objects, and Attributes

Figure 3 gives the Z model of HLA classes and class attributes that will be used as the basis for all further descriptions. The first line introduces two basic kinds of entities: classes and class attributes. The axiomatic definition describes two functions and a constraint between them that must always be maintained. The *AttributesToClass* function maps every class attribute to a single class.

Within the HLA, the ability to delete an object is obtained by becoming the owner of the special attribute, called the *privilegeToDeleteObject* attribute, for that object. Although

$[OBJECT, OBJECTATTR]$

$$
\begin{array}{|l}
\hline
\textit{ObjectCollection} \\
\textit{Objects} : \mathbb{P}\ \textit{OBJECT} \\
\textit{ObjectsToClass} : \textit{OBJECT} \rightarrow \textit{CLASS} \\
\textit{ObjectAttrs} : \mathbb{P}\ \textit{OBJECTATTR} \\
\textit{ObjectAttrToObject} : \textit{OBJECTATTR} \twoheadrightarrow \textit{OBJECT} \\
\textit{ObjectAttrToClassAttr} : \textit{OBJECTATTR} \rightarrow \textit{CLASSATTR} \\
\hline
\textit{ObjectAttrs} = \mathrm{dom}\ (\textit{ObjectAttrToObject} \rhd \textit{Objects}) \\
\textit{ObjectToClass}\ ;\ \textit{AttributesToClass}{\sim} = \textit{ObjectAttrToObject}{\sim}\ ;\ \textit{ObjectAttrToClassAttr} \\
(\ \textit{ObjectAttrToObject}\ ;\ \textit{ObjectAttrToObject}{\sim}) \cap \\
\qquad\qquad (\ \textit{ObjectAttrToClassAttr}\ ;\ \textit{ObjectAttrToClassAttr}{\sim}) \\
\qquad \subseteq \mathrm{id}\ \textit{OBJECTATTR} \\
\hline
\end{array}
$$

Figure 4: Z model of objects and object attributes

privilegeToDeleteObject is a fully defined class attribute, it is expected that federates will rarely associate a value with it. This special attribute, which must be defined for every class, is modeled by the *privToDeleteObject* function in the Z model. The constraint requires that the privilege to delete attribute that is specially denoted for a class must be a class attribute for that class.

Figure 4 gives the Z description of objects in HLA. The schema *ObjectCollection* introduces two variables describing HLA objects: *Objects*, the set of objects currently recognized by the RTI, and *ObjectsToClass*, the mapping that identifies the class associated with each object. The set *ObjectAttrs* contains all the object attributes related to the currently known objects, as required by the first state invariant. The two projection functions, *ObjectAttrToObject* and *ObjectAttrToClassAttr*, relate object attributes back to the corresponding objects and class attributes. As indicated by its double-headed arrow, *ObjectAttrToObject* is a function that maps onto its range: that is, every object has at least one object attribute, the one associated with the *privToDeleteObject* class attribute.

The final two state invariants define the required correspondence between object attributes, objects, and class attributes. The first of these constraints specifies that for any object and any class attribute defined by that object's class, a corresponding object attribute relates the object and the class attribute. The final constraint specifies that no two object attributes relate the same object and class attribute.

There is an alternative formulation of the object attribute construct. As shown in Figure 5, each object attribute could be viewed as an ordered pair, with the complete collection of object attributes constructed directly from the existing variables. However, the Ladybug input language NP does not support denoting a particular object attribute in this formulation, so we chose to use the otherwise more cumbersome representation shown in Figure 4.

$$ObjectAttrs : \mathbb{P}\,(OBJECT \times CLASSATTR)$$

$$\underline{ObjectAttrs = Objects} \lhd ObjectToClass \mathbin{;} AttributesToClass^{\sim}$$

Figure 5: Alternative model of object attributes

2.2 Required State

The simulation state, as shown in Figure 6, describes the state of the simulation that is explicitly described in the IFSpec. We have chosen to separate the explicit state from the implicit state (represented by the internal state given in Figure 7) for three reasons:

- Ease of validation. Because the simulation state is explicitly described in the IFSpec, it is easily checked against the informal specification (the IFSpec). The implicit state, on the other hand, requires significantly more effort to check against the original specification. By culling it out separately, the original specification writers are likely to pay closer attention to the implicit state.
- Isolation for analysis. Some claims require only the explicit state. By separating the implicit state, we reduce the number of cases required for Ladybug to check for these claims.
- Implementation freedom. The simulation state must be faithfully implemented in any actual code. Although the behavior of the implicit state is required in some form, the implementors have more freedom to choose an alternative structuring of this information in the final design.

The *SimulationState* schema introduces three new variables, but no new constraints. *Federates* is the set of federates currently joined in the simulation. *Publishing* and *Owns* describe the attributes published and owned by each federate, as described in the IFSpec. A full model would also

$[FEDERATE]$

SimulationState

$ObjectCollection$
$Federates : \mathbb{P}\ FEDERATE$
$Publishing : FEDERATE \leftrightarrow CLASSATTR$
$Owns : FEDERATE \leftrightarrow OBJECTATTR$

Figure 6: Model of (explicit) simulation state.

introduce a variable describing the subscribe relations, but subscribing is irrelevant to the properties that we will be checking and has been omitted.

Figure 7 lists the schema *OwnershipInternalState*, which models the implicitly described state. This schema includes two variables that indicate each federate's willingness to accept or divest ownership of a specific object attribute. These variables were introduced based on statements in the IFSpec such as

> The federate has informed the RTI of its intent to divest ownership of the specified attributes.

which appears in the post-condition of the description of the *Request Attribute Ownership Divestiture* service (see Figure 1). This state also records the set of federates that may gain ownership of an object attribute as indicated by the *Request Attribute Ownership Divestiture* service.

OwnershipInternalState
$WillingToDivest : FEDERATE \leftrightarrow OBJECTATTR$
$WillingToAccept : FEDERATE \leftrightarrow OBJECTATTR$
$TargetOwners : FEDERATE \leftrightarrow OBJECTATTR$

Figure 7: Model of (implicit) internal state.

For convenience, we combine the explicit state and the implicit state into a single schema, *ExecutionState*.

ExecutionState
SimulationState
OwnershipInternalState

Figure 8: Model of complete required state.

2.3 Two Properties: NoTwoOwners and CompleteOwners

We specified eight properties in the full Z model. In this section, we describe two of these properties in detail, *NoTwoOwners* and *CompleteOwners*.

We model all the properties as constraints on the state, including either the explicit state or the implicit state, or both. By describing the properties separately from the base description, we can pose questions about whether the system does or does not have a specified property. Ladybug can also check whether operations (or sequences of operations) maintain these properties.

The schema *NoTwoOwners*, given in Figure 9, is based on the IFSpec statement

> The privilege to update a value for an attribute is uniquely held by a single federate at any given time during a federation execution.

This property depends only on the explicit state that is described by the schema *SimulationState*. The condition on this property requires the inverse of *Owns* to be a function from *OBJECTATTR* to *FEDERATE*, implying that *Owns* itself is injective. This, in turn, implies that every object

NoTwoOwners
SimulationState

$Owns^{\sim} \in (OBJECTATTR \rightarrowtail FEDERATE)$

Figure 9: Z model of unique ownership property.

attribute is owned by no more than one federate. The designers of the HLA view this as a property an invariant.

The second property detailed here, *CompleteOwners*, requires that every object attribute be owned by some federate. Figure 10 lists the Z model of *CompleteOwners*. Unlike most of the properties modeled, *CompleteOwners* is not required by the IFSpec and is not an invariant, as some services (including unconditional divestitures, unpublishing class attributes, and resigning from the federation) may leave object attributes unowned. However, this property is still worth considering as it should be invariant across some complete protocols, such as negotiated divestiture and acquisition. This property is checked by verifying that if every attribute is owned when a protocol begins (and there are no other concurrent services), then every attribute is owned when the protocol finishes.

CompleteOwners
SimulationState

$ran\ Owns = ObjectAttrs$

Figure 10: Z model of universal ownership property.

2.4 Two Operations: RequestAttrOwnDivestiture and AttrOwnDivestNotify

Of the ten total operations specified in the full Z model of ownership management, we detail two operations, *RequestAttrOwnDivestiture* and *AttrOwnDivestNotify*. These operations combine to implement the simplest unconditional divestiture protocol execution.

The model of these operations, as with all others, consists of three pieces: the arguments, the pre-conditions, and the post-conditions. We follow the Z convention, appending a question mark (?) to the end of the name of each input parameter. As detailed below, we have chosen to model only the subset of the parameters that is relevant to our analysis. We translate the pre-conditions into state invariants on the pre-state, again ignoring pre-conditions irrelevant to our needs. We similarly translate the post-conditions as state invariants of the post-state (as indicated by the primed variables).

The *Request Attribute Ownership Divestiture* service, which is described informally in Figure 1, allows a federate to notify the RTI that it (the federate) no longer wishes to be responsible for updating any of a set of object attributes. When the RTI responds with an invocation of the *Attribute Ownership Divestiture Notification*, the originating federate is no longer responsible for the object attributes in question.

$$
\begin{array}{|l}
\textit{RequestAttrOwnDivestiture} \\
\Delta \textit{ExecutionState} \\
\textit{fed}? : \textit{FEDERATE} \\
\textit{targets}? : \mathbb{P}\ \textit{FEDERATE} \\
\textit{obj}? : \textit{Object} \\
\textit{cattrs}? : \mathbb{P}\ \textit{CLASSATTR} \\
\textit{oattrs} : \mathbb{P}\ \textit{OBJECTATTR} \\
\hline
\textit{ObjectAttrToClassAtttr}(\textit{oattrs}) = \textit{cattrs}? \\
\textit{ObjectAttrToObject}(\textit{oattrs}) = \{\textit{obj}?\} \\
\textit{fed}? \in \textit{Federates} \\
\{\textit{fed}?\} \times \textit{oattrs} \subseteq \textit{Owns} \\
\textit{WillingToDivest}' = \textit{WillingToDivest} \cup (\ \{\textit{fed}?\} \times \textit{oattrs}) \\
\textit{WillingToAccept}' = \textit{WillingToAccept} \\
\textit{TargetOwners}' = \textit{TargetOwners} \cup (\textit{targets}? \times \textit{oattrs}) \\
\textit{SimulationState}' = \textit{SimulationState}
\end{array}
$$

Figure 11: The Z model of the *Request Attribute Ownership Divestiture* service.

Figure 11 shows the model of the Request Attribute Ownership Divestiture service. This operation requires all the execution state, including the implicit state described by *OwnershipInternalState*. The operation takes four inputs, *fed*?, the federate seeking to relinquish ownership, *targets*?, the set of potential new owners, *obj*?, the object whose attributes are being disowned, and *cattrs*?, a set of class attributes describing the object attributes to be disowned.

The IFSpec states that the *targets*? parameter is optional, but Z does not directly support optional parameters. To handle this complication, we assume that the set of all *Federates* is passed when no constraint is requested. We have also chosen to ignore two of the actual arguments specified in the IFSpec. The user-supplied tag, although important in any actual implementation, does not affect any interesting properties. The conditional divestiture flag is ignored here because it represents control flow, rather than the resulting structure. The model does not require divestiture (or disallow unconditional divestiture), so the analysis will consider both cases of the flag.

The first four conditions capture the relevant pre-conditions, whereas the final four conditions capture the post-conditions. The first two conditions assert that the set of object attributes, referred to by *oattrs*, is the set matching the object, referred to by *obj?*, and the class attributes, referred to by the set *cattrs?*. The third condition, $\textit{fed}? \in \textit{Federates}$, enforces the second pre-condition in the IFSpec

> The federate has joined the federation execution.

The fourth condition, $\{\textit{fed}?\} \times \textit{oattrs} \subseteq \textit{Owns}$, enforces the IFSpec pre-condition

> The federate owns the specified attributes.

The next three conditions describe the change to the internal state. After the request, the federate is willing to divest the indicated object attributes, but there is no change in any federate's

$$
\begin{array}{l}
\textbf{AttrOwnDivestNotify} \\
\hline
\Delta ExecutionState \\
fed? : FEDERATE \\
obj? : Object \\
cattrs? : \mathbb{P}\ CLASSATTR \\
oattrs : \mathbb{P}\ OBJECTATTR \\
\hline
ObjectAttrToClassAtttr(oattrs) = cattrs? \\
ObjectAttrToObject(oattrs) = \{obj?\} \\
fed? \in Federates \\
\{fed?\} \times oattrs \subseteq Owns \\
\\
Owns' = Owns \setminus (\{fed?\} \times oattrs) \\
Objects' = Objects \\
Publishing' = Publishing \\
ObjectAttrs' = ObjectAttrs \\
Federates' = Federates \\
\\
WillingToAccept' = WillingToAccept \\
WillingToDivest' = WillingToDivest \setminus (\{fed?\} \times oattrs) \\
TargetOwners' = TargetOwners \\
\hline
\end{array}
$$

Figure 12: The Z model of the *Attribute Ownership Divestiture Notify* service.

willingness to accept new ownership. This change, modeled by the fifth condition, is required by the IFSpec post-condition

> The federate has informed the RTI of its request to divest ownership of the specified attributes.

We assume that the willingness to accept ownership does not change due to this operation, as modeled by the next condition. This assumption, although reasonable and the likely intent of the specifiers, indicates a lacking in the original specification and is closely related to other assumptions described later in this section.

The target owners to consider, as described by the *targets?* parameter, are recorded, supporting option 1 in the IFSpec.

> The federate can specify which federate(s) can take ownership of the released attributes, otherwise any federate may own them.

The final condition, $SimulationState' = SimulationState$, captures the IFSpec post-condition

> No change in attribute ownership.

Figure 12 shows the response service from the RTI. The *AttrOwnDivestNotify* operation takes four arguments, similar to those used as inputs to *ReqAttrOwnDivest* operation. The operation requires that the federate being notified is currently a member of the federation and owns the object attributes in question.

After the operation, the ownership has changed, with the target federate no longer owning the target object attributes. In addition, the *WillingToDivest* relation is updated, removing the pending desire to divest (which has now been fulfilled). This latter change is not specified in the IFSpec. In fact, version 1.2 of the IFSpec never states when a willingness to divest (or accept) should be cancelled (or maintained). Based at least partially on our feedback, version 1.3 of the IFSpec [DoD98] does state when these intentions should be cancelled. To progress in the analysis, we chose to specify "reasonable" points for cancelling the intentions.

In draft 4 of version 1.2 (no longer available on the web), the IFSpec placed no pre-condition on the Attribute Ownership Divestiture Notification service requiring that the federate had previously attempted to divest ownership of the specified attributes. Draft 6 of version 1.2 partially repairs this flaw with the pre-condition

> A federate has previously attempted to divest ownership of the specified attributes

This pre-condition is still flawed. It should require that the federate that currently owns the attribute has requested to divest ownership, without any subsequent cancellation of its willingness to divest.

It should not come as a surprise that both of these inconsistencies (as well as other, similar ones discovered with other services) arise in conjunction with the state that has been only implicitly specified.

We also discovered an ambiguity during this formalization. The IFSpec does not indicate if the RTI is allowed to satisfy the divestiture partially. It is similarly unclear if the RTI may combine multiple ownership divestiture requests, returning a single divestiture notification. We have assumed in our model that both possibilities are allowable, but some conforming implementations may disallow one or both variations.

3. Analyzing the Formal Model

The final step in this process is to analyze the model produced with our chosen tool, Ladybug. This analysis consists of three steps, detailed in the following sections: translating the Z model to

```
/* Define the basic universe */
ObjectCollection = [
  Objects: set OBJECT
  Object_Attrs: set OATTR
  ObjectToClass: tot OBJECT -> CLASS
  ClassAttrsToClass: tot ATTR -> CLASS
  ObjAttrsToClassAttrs: tot OATTR -> ATTR
  ObjAttrsToObject: tot suj OATTR -> OBJECT
|
   /* Only object attributes about known objects are of interest */
   Object_Attrs = dom (ObjAttrsToObject :> Objects)
]


GoodObjColl = [
  ObjectCollection
|
  ObjectToClass;ClassAttrsToClass~ =
      ObjAttrsToObject~;ObjAttrsToClassAttrs
  ObjAttrsToObject;ObjAttrsToObject~ &
      ObjAttrsToClassAttrs;ObjAttrsToClassAttrs~ <= Id
/*  First invariant says that each instance has the attributes
specified by its class (or has the right number of attributes).
second invariant states that the intersection of the
two equivalence relations on ObjAttrToObject and ObjAttrsToClassAttrs
intersect only when the same object attributes
are the subject, i.e., two object attributes can't be of the
same type and belong to the same object instance.  */
]
```

Figure 13: The NP schemas ObjectCollection and GoodObjColl

Ladybug's input language NP, constructing claims about the model to be checked, and reviewing the results of checking the claims with Ladybug.

3.1 Translating the Z Model

In order to check the specification, Ladybug requires the specification to be written in its input language, NP. The translation from Z to NP is generally straightforward, mostly consisting of transliterating special Z symbols into the equivalent NP ASCII constructs. Only a few items within the translation are worth noting specifically. The complete NP specification is given in Appendix C.

Figure 13 shows the ObjectCollection schema, translated from the Z schema *ObjectCollection* given in Figure 4. NP does not the support the axiomatic definitions used in Figure 3 (only schemas), so these initial definitions have been merged into the `ObjectCollection` schema. The *privilegeToDeleteObject* attribute is never used by any of the properties considered, so we have not introduced any equivalent of the *privToDeleteObject* variable into the NP specification.

We separated two of the conditions in *ObjectCollection* into a separate schema, called `GoodObjColl`. As noted earlier, this separation enables these properties to be checked. Because

```
/* Explicitly defined state */
SimState = [
  GoodObjColl
  Federates: set FED
  Publishing: FED <-> ATTR
  Owns: FED <-> OATTR
]

/* Implicitly defined state */
OwnershipInternalState = [
  WillingToDivest:FED <-> OATTR
  WillingToAccept: FED <-> OATTR
  TargetOwners : FED <-> OATTR
]

/* Total state to consider */
ExecutionState =
[SimState
OwnershipInternalState]
```

Figure 14: The NP specification of the explicit, implicit, and total state.

all the later properties we will check assume a sound `ObjectCollection`, the `GoodObjColl` schema, rather than the raw `ObjectCollection` schema, is imported into `SimState`. Figure 14 shows the NP translation of the explicit state into the schema `SimState`, as well as the implicit and total state.

To simplify the analysis, we model the operations as directly accepting a set of object attributes, rather than the actual set of class attributes. We feel that explicitly specifying the requirement to translate from class attributes to object attributes is important in the formal specification, as this is a possible stumbling block in an actual implementation. However, a faulty translation from class attributes to object attributes is a flaw in the implementation, not the overall HLA design. This

```
RequestAttrOwnDivestiture(fed?:FED, obj?:OBJECT, targets?:set FED,
                                          oattrs?:set OATTR) =
[
  ExecutionState
  const SimState
|
  fed? in Federates
  ObjAttrsToObject.oattrs? = {obj?}
  /* ({fed?} <: Un :> oattrs?) is the same as Z {fed?} x oattrs */
  ({fed?} <: Un :> oattrs?) <= Owns

  WillingToDivest' = WillingToDivest U ({fed?} <: Un :> oattrs?)
  WillingToAccept' = WillingToAccept
  TargetOwners' = TargetOwners U (targets? <: Un :> oattrs?)
]
```

Figure 15: The NP specification of the *Request Attribute Ownership Divestiture* service.

```
/* Check if the non-empty state allows two owners */

NoTwoOwners = [NonEmpty | fun Owns~]

NoBadOwnedAttrs  =  [SimState | ran Owns = Object_Attrs]

NoBadOwners = [SimState | dom Owns <= Federates]

OwnsOnlyIfPublishes = [SimState | Owns;ObjAttrsToClassAttrs <=
Publishing ]

SoundOwners = [
NoTwoOwners
NoBadOwnedAttrs
NoBadOwners
OwnsOnlyIfPublishes]


AttrDivNotSoundOwns(fed:FED, obj:OBJECT, oattrs:set OATTR)::
   SoundOwners and AttrOwnDivestNotify(fed,obj,oattrs) => SoundOwners'
```

Figure 16: The NP sound ownership properties and the NP claim that the sound ownership properties are invariant across the Attribute Ownership Divestiture Notify service.

analysis is attempting to check the design, so this complication is unnecessary in the NP translation. Removing this complication both makes the NP specification easier to read and reduces the number of cases to be considered by the checker Ladybug. Figure 15 shows the NP operation `RequestAttrOwnDivestiture`.

Because NP does not directly support cross products, some of the conditions within the operation definitions need to be recast slightly. The Z expression

$$\{fed?\} \times oattrs$$

can be translated to the NP expression

```
{fed?} <: Un :> oattrs?
```

where `Un` is the universal relation, forced by type to be `FEDERATE <-> OATTR`.

3.2 Constructing the Claims

We have constructed two kinds of claims about the ownership management specification. The simpler claims assert that a property is invariant across any possible invocation on a single operation. The more complicated claims describe an entire protocol execution, asserting that some property holds after the entire execution if it holds prior to the execution.

Figure 16 shows a simple operation invariant claim written in NP. Unlike schemas in NP, which use an equals sign (=) to separate the header of the schema from its body, claims in NP separate the header from the body with a double colon (::). The `AttrDivNotSoundOwns` claim asserts that for any federate, object, and set of object attributes, the properties described in the schema `SoundOwners` (which requires unique valid ownership, but not universal ownership), is invariant across the *Attribute Ownership Divestiture Notify* service (as described in the NP schema

```
/* Check for complete ownership after a simple conditional divestiture */
ConditionalCompleteOwners(fed1:FED, fed2:FED, targets : set FED,
                          obj:OBJECT, oattrs1:set OATTR,
                          oattrs2:set OATTR)::
[
  ExecutionState
|
  /* require the case we are interested in */
  not fed2 = fed1 and
  fed2 in targets and
  oattrs2 <= oattrs1 and
  SoundOwners and
  CompleteOwners and
  /* conditional divestiture of oattrs1, actually divesting oattrs2 */
     (RequestAttrOwnDivestiture(fed1,obj,targets,oattrs1);
      RequestAttrOwnAssumption(fed2,obj,oattrs1);
      RequestAttrOwnAcquisition(fed2,obj,oattrs2);
      AttrOwnDivestNotify(fed1,obj,oattrs2);
      AttrOwnAcquisitionNotify(fed2,obj,oattrs2)) =>
  CompleteOwners'
]
```

Figure 17: The NP claim asserting that all object attributes are owned after a conditional divestiture.

`AttrOwnDivestNotify`), using that federate, object, and set of object attributes as the arguments to `AttrOwnDivestNotify`.

The remaining claims are more complex, as they check properties that are not invariants, but should hold at specified points during the protocol. We do not recheck properties that have been shown invariant across all operations, as they must also hold invariant across any combination of operations that comprise a complete protocol. Figure 17 shows the NP claim that asserts that a conditional divestiture protocol does not lose ownership of objects.

The five indented lines near the end of the claim describe one possible sequence of services for this protocol. A federate (`fed1`) initiates the protocol by requesting conditional divestiture of a set of object attributes (`oattrs1`). The RTI then requests that a second federate (`fed2`) assume ownership of those attributes. The second federate agrees to take ownership of a subset of the attributes (`oattrs2`) and requests that ownership from the RTI. The RTI can then respond to the first federate with a divestiture notification of the subset of attributes. Finally, the RTI grants ownership of a subset of the object attributes divested to the second federate.

3.3 The Analysis

Analyzing the claims using Ladybug is nearly automatic. The only significant choice left to the analyst at this stage is to bound the number of elements to be considered by Ladybug in the analysis. Ladybug only searches for counter-examples in a limited finite space. The *scope* limits how many instances of each given type (such as `FED` or `OATTR`) should be considered.

The default scope assumes three elements of each type, which suffices for many specifications. For the ownership management specification, however, we chose to vary the scope from the default to gain more confidence in the analysis. For these claims, we chose to limit the number of classes to only one, as class distinctions are irrelevant to our concerns here. We limited the number of federates and class attributes to two apiece, permitting potentially divided ownership of a single object. We limited the number of objects to three. These restrictions in turn require the support of six object attributes (three objects with two object attributes apiece). When choosing the scope, interactions such as this must be carefully noted if the analysis is to be trusted. Limiting the number of object attributes to fewer than six would force the analysis to consider fewer than three objects in order to satisfy the other requirements.

The Ladybug analysis was completed quickly using both the default and selected scopes. Most of the checks required a few seconds to complete (when run on a 400Mhz Pentium II using the Sun JDK 1.1.6). The complete protocol executions required more time to check with the same scope (ranging from ten minutes to over a day), so we have chosen a different scope to check these claims, reducing their check time to a few seconds.

The first issue arising from the analysis involves the set of federates joined in the federation. The IFSpec never explicitly states that this set is unchanged by the execution of these operations, although the clear intent is that the only means that a federate can leave the federation is through the use of the *Resign Federation Execution* service. We adjusted the NP specification to capture this requirement, which should also be propagated back to the IFSpec.

With that omission repaired, we again analyzed the specification. As indicated by the output shown in Figure 18, the Ladybug analysis found that the *Attribute Ownership Acquisition Notify* service does not hold the sound ownership properties invariant. In particular, the RTI may grant ownership of an object attribute to a federate that is not publishing the corresponding class attribute. This can most clearly by seen by noticing that the `Publishing'` relationship is empty, indicating that no class attributes are being published and therefore no object attributes can be owned.

Reviewing the IFSpec, the only relevant pre-condition for this service is

> The federate has previously attempted to acquire ownership of the attribute.

Publishing the corresponding class attribute is a pre-condition of requesting ownership, so this combination might be expected to hold the invariant against any actual protocol execution. However, if the federate unpublishes the class after requesting ownership, but prior to being granted ownership, a condition occurs where a federate can be granted ownership of an object attribute while not publishing the corresponding class attribute. An instance of this problem can be shown with the counterexample generated for the `UnpublishInAcquisition` claim.

Version 1.3 of the IFSpec adds the pre-condition

> The federate is publishing the corresponding class attributes at the known class of the specified object instance.

An alternative solution to this inconsistency is to have the unpublish operation cancel the federate's willingness to acquire any related object attributes.

Table 1 summarizes the results of the checks made with the non-default scopes. The columns listing the names of given types indicate the scope limit given for those types. The times given are in seconds. The entire state space was checked for all claims but the case presenting a

```
Found Counterexample to Claim
AttrAcqNotSoundOwns:
ClassAttrsToClass : tot ATTR->CLASS =
  { a0 -> c0,
   a1 -> c0 }
ClassAttrsToClass' : tot ATTR->CLASS =
  { a0 -> c0,
   a1 -> c0 }
fed : FED =
  f0
Federates : set FED =
  { f0 }
Federates' : set FED =
  { f0 }
oattrs : set OATTR =
  { oa0 }
obj : OBJECT =
  ob0
ObjAttrsToClassAttrs : tot OATTR->ATTR =
  { oa0 -> a0,
   oa1 -> a1,
   oa2 -> a0,
   oa3 -> a1,
   oa4 -> a0,
   oa5 -> a1 }
ObjAttrsToClassAttrs' : tot OATTR->ATTR =
  { oa0 -> a0,
   oa1 -> a1,
   oa2 -> a0,
   oa3 -> a1,
   oa4 -> a0,
   oa5 -> a1 }
ObjAttrsToObject : tot OATTR->OBJECT =
  { oa0 -> ob0,
   oa1 -> ob0,
   oa2 -> ob1,
   oa3 -> ob1,
   oa4 -> ob2,
   oa5 -> ob2 }
ObjAttrsToObject' : tot OATTR->OBJECT =
  { oa0 -> ob0,
   oa1 -> ob0,
   oa2 -> ob1,
   oa3 -> ob1,
   oa4 -> ob2,
   oa5 -> ob2 }
Object_Attrs : set OATTR =
  { oa0, oa1 }
Object_Attrs' : set OATTR =
  { oa0, oa1 }
Objects : set OBJECT =
  { ob0 }
Objects' : set OBJECT =
  { ob0 }
ObjectToClass : tot OBJECT->CLASS =
  { ob0 -> c0,
   ob1 -> c0,
   ob2 -> c0 }
ObjectToClass' : tot OBJECT->CLASS =
  { ob0 -> c0,
   ob1 -> c0,
   ob2 -> c0 }
Owns : FED<->OATTR =
  {  }
Owns' : FED<->OATTR =
  { f0 -> {oa0 } }
Publishing : FED<->ATTR =
  {  }
Publishing' : FED<->ATTR =
  {  }
TargetOwners : FED<->OATTR =
  { f0 -> {oa0 } }
TargetOwners' : FED<->OATTR =
  {  }
WillingToAccept : FED<->OATTR =
  { f0 -> {oa0 } }
WillingToAccept' : FED<->OATTR =
  {  }
WillingToDivest : FED<->OATTR =
  {  }
WillingToDivest' : FED<->OATTR =
  {  }
```

Figure 18: Output demonstrating a counterexamples discovered by Ladybug to the claim AttrAcqNotSoundOwn.

Claim	Description	ATTR	CLASS	FED	OATTR	OBJ	Time
ReqAttrDivSoundOwns	Check that the *Request Attribute Divestiture* service maintains sound ownership.	2	1	2	6	3	7.1
ReqAttrAcqSoundOwns	Check that the *Request Attribute Acquisition* service maintains sound ownership.	2	1	2	6	3	19.8
AttrDivNotSoundOwns	Check that the *Attribute Divestiture Notification* service maintains sound ownership.	2	1	2	6	3	3.2
AttrAcqNotSoundOwns	Check that the *Attribute Acquisition Notification* service maintains sound ownership.	2	1	2	6	3	1.2*
PublishSoundOwns	Check that the *Publish Object Class* service maintains sound ownership.	2	1	2	6	3	2.6
UnpublishSoundOwns	Check that the *Unpublish Object Class* service maintains sound ownership.	2	1	2	6	3	2.2
ConditionalCompleteOwners	Check that a simple conditional divestiture protocol execution maintains complete ownership.	3	1	2	3	1	0.1
UnconditionalSoundTargets	Check that a simple unconditional divestiture protocol execution leaves all targets unowned.	3	1	2	3	1	0.1
ConditionalSoundTargets	Check that a simple conditional divestiture protocol execution leaves the targets set sound.	3	1	2	3	1	4.4

Table 1: Summary of the checks done by Ladybug. The time for `AttrAcqNotSoundOwns` is the time required to find the first counterexample.

counterexample, where the search was halted after the first counterexample was found. All checks were made using the default Ladybug settings (except for scope).

The complete output from the non-default scope Ladybug runs is given in Appendix D.

4. Conclusions

We discovered several inconsistencies and ambiguities during the formalization and analysis of an informal specification. While generally minor in nature, these flaws could introduce significant difficulties into the HLA development, if not caught at design time. Components, being developed by disparate organizations with possibly disparate interpretations of the IFSpec, could fail when joined together, forcing expensive testing and re-writes. With help from our feedback, these issues have been resolved in a new version of the IFSpec [DoD98].

Not surprisingly, most of the issues uncovered involved the implicitly-specified state. In our experience, lack of detailed consideration of portions of the system leads to many of the flaws in system designs. Informal specifications facilitate this lack of consideration by allowing seemingly obvious portions of the system to remain unspecified or implicitly specified. Formalization helps identify these missing pieces.

In addition to the two counterexamples discovered, Ladybug made two notable contributions to the analysis:

- As a forcing function. Issues with what properties should be checked in the specification and how to check those properties in the specification force additional consideration of the problem that an alternative formalization might have missed. In particular, the problems with mismatch in the set of object attributes is more apparent when attempting to define specific protocol executions.
- Increased assurance of the correctness of the design. A lack of flaws is the eventual goal of any design. Assurance that at least selected possible flaws are not present is a significant first step. Experiences with analysis of other specifications, such as the new Mobile IPv6 standard [JNW98], have shown that flaws in systems can be discovered using a tool similar to Ladybug.

However, any automated analysis tool has fundamental limitations that should also be kept in mind:

- Only properties explicitly described are checked. Many flaws that have not been considered may remain. Ladybug cannot *generate* interesting claims, but rather can only *check* claims made by the analyst.
- The structure of the specification may hide flaws allowed by the design. As an example, the structure of the Z model requires that the services be treated atomically, with no concurrent interaction. With a distributed system such as HLA, such interactions are likely, leaving possible flaws undetected. Performing analyses of the same system with multiple tools and formalisms can help reduce these holes.
- The specification itself may be consistent, but it may not correctly capture the intent of the designers. Actual implementations, developed by humans who may understand that intent, may introduce flaws present in the design, but not captured in the formal specification.
- Finite checkers, such as Ladybug, place bounds on the problem to enable analysis. Flaws may exist only in systems which exceed those artificial bounds. Although we have yet to find a flaw in a design missed by a reasonable scope in any of our analyses, such flaws certainly exist in at least some designs.

In summary, our formalization and subsequent analysis discovered some flaws in the IFSpec and achieved a reasonable level of assuredness that other potential flaws are not present.

Additional analysis of these specifications are possible. Many other protocol sequences are meaningful and could be checked. Manual generation of these protocol executions is tedious and time-consuming. Due to time constraints, we have chosen at this time to only investigate a handful of these possible protocols. An ideal analysis tool could consider both a CSP specification, which can express the possible protocol executions succinctly, and a Z specification, which can express the outcomes of a particular protocol execution succinctly, and thus automate this task. One possible future path being considered is the generation of interesting executions using a CSP checker (FDR [FDR97]), with a manual, or possibly automated, conversion of the output into NP claims.

5. Bibliography

[AGI98] Robert J. Allen, David Garlan, and James Ivers. Formal Modeling and Analysis of the HLA Component Integration Standard, *Proceedings of the Sixth International Symposium on the Foundations of Software Engineering* (FSE-6), November 1998.

[CW+96] Edmund M. Clarke, Jeanette M. Wing, et al. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys,* Vol. 28, No. 4, December 1996, pp. 626–643.

[DoD97] Defense Modeling and Simulation Office. *High Level Architecture Interface Specification Version 1.2*. August 13, 1997. http://www.dmso.mil/projects/hla/tech/if-spec.

[DoD98] Defense Modeling and Simulation Office. *High Level Architecture Interface Specification Version 1.3*. April 20, 1998. http://www.dmso.mil/projects/hla/tech/if-spec.

[FDR97] *Failures Divergence Refinement: FDR2 User Manual*, version 2.22. Formal Systems (Europe) Ltd, Oxford, England, October 1997.

[Hoa85] Hoare, C.A.R. *Communicating Sequential Processes*. Prentice Hall, 1985.

[JD96a] Daniel Jackson and Craig A. Damon. *Nitpick: A Checker for Software Specifications (Reference Manual).* Technical Report CMU-CS-96-109, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, January 1996.

[JD96b] Daniel Jackson and Craig A. Damon. Elements of Style: Analyzing a Software Design Feature with a Counterexample Detector. *IEEE Transactions on Software Engineering,* July 1996, Vol. 22, No. 7, pp. 484–495.

[JNW98] Daniel Jackson, Yuchang Ng, and Jeannette M. Wing. *A Nitpick Analysis of Mobile IPv6*. Technical Report CMU-CS-98-113, Carnegie Mellon University, Pittsburgh, PA, March, 1998.

[Spi92] J. M. Spivey. *The Z Notation: A Reference Manual,* Second edition, Prentice Hall, 1992.

Appendix A. Full Z Description of the HLA Specification.

[*CLASS*,*CLASSATTR*,*OBJECT*, *OBJECTATTR*, *FEDERATE*]

$$
\begin{array}{|l}
AttributesToClass : CLASSATTR \rightarrow CLASS \\
privToDeleteObject : CLASS \rightarrowtail CLASSATTR \\
\hline
privToDeleteObject^{\sim} \subseteq AttributesToClass
\end{array}
$$

ObjectCollection

$$
\begin{array}{|l}
Objects : \mathbb{P}\ OBJECT \\
ObjectsToClass : OBJECT \rightarrow CLASS \\
ObjectAttrs : \mathbb{P}\ OBJECTATTR \\
ObjectAttrToObject : OBJECTATTR \twoheadrightarrow OBJECT \\
ObjectAttrToClassAttr : OBJECTATTR \rightarrow CLASSATTR \\
\hline
ObjectAttrs = \mathrm{dom}\ (ObjectAttrToObject \rhd Objects) \\
ObjectToClass \ ;\ AttributesToClass^{\sim} = ObjectAttrToObject^{\sim} \ ;\ ObjectAttrToClassAttr \\
(\ ObjectAttrToObject \ ;\ ObjectAttrToObject^{\sim}) \cap \\
\qquad\qquad (\ ObjectAttrToClassAttr \ ;\ ObjectAttrToClassAttr^{\sim}) \\
\qquad \subseteq\ \mathrm{id}\ OBJECTATTR
\end{array}
$$

SimulationState

$$
\begin{array}{|l}
ObjectCollection \\
Federates : \mathbb{P}\ FEDERATE \\
Publishing : FEDERATE \leftrightarrow CLASSATTR \\
Owns : FEDERATE \leftrightarrow OBJECTATTR
\end{array}
$$

OwnershipInternalState

$$
\begin{array}{|l}
WillingToDivest : FEDERATE \leftrightarrow OBJECTATTR \\
WillingToAccept : FEDERATE \leftrightarrow OBJECTATTR \\
TargetOwners : FEDERATE \leftrightarrow OBJECTATTR
\end{array}
$$

ExecutionState

$$
\begin{array}{|l}
SimulationState \\
OwnershipInternalState
\end{array}
$$

NoTwoOwners

SimulationState

$$Owns^{\sim} \in (OBJECTATTR \rightarrowtail\!\!\!\!\!\!\!\!\rightarrow FEDERATE)$$

NoBadOwnedAttrs

SimulationState

$$\mathrm{ran}\ Owns \subseteq ObjectAttrs$$

NoBadOwners

SimulationState

$$\mathrm{dom}\ Owns \subseteq Federates$$

OwnsOnlyIfPublishes

SimulationState

$$(Owns \mathbin{;} ObjectAttrToClassAttr) \subseteq Publishing$$

CompleteOwners

SimulationState

$$ran\ Owns = ObjectAttrs$$

SoundDivestments

ExecutionState

$$WillingToDivest \subseteq Owns$$

SoundAccepts

ExecutionState

$$WillingToAccept \cap Owns = \emptyset$$

TargetsUnowned

ExecutionState

$$\mathrm{ran} TargetOwners \cap \mathrm{ran}\ Owns = \emptyset$$

$$
\begin{array}{|l}
\hline
\textit{CreateFedExecution} \\
\textit{ExecutionState}' \\
\hline
\textit{Objects}' = \varnothing \\
\textit{Federates}' = \varnothing \\
\textit{Publishing}' = \varnothing \\
\textit{Owns}' = \varnothing \\
\\
\textit{WillingToAccept}' = \varnothing \\
\textit{WillingToDivest}' = \varnothing \\
\textit{TargetOwners}' = \varnothing \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{JoinFedExecution} \\
\Delta \textit{ExecutionState} \\
\textit{fed}? : \textit{FEDERATE} \\
\hline
\textit{fed}? \notin \textit{Federates} \\
\\
\textit{Federates}' = \textit{Federates} \cup \{\textit{fed}?\} \\
\textit{ObjectCollection}' = \textit{ObjectCollection} \\
\textit{OwnershipInternalState}' = \textit{OwnershipInternalState} \\
\textit{Publishing}' = \textit{Publishing} \\
\textit{Owns}' = \textit{Owns} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{RequestAttrOwnDivestiture} \\
\Delta \textit{ExecutionState} \\
\textit{fed}? : \textit{FEDERATE} \\
\textit{targets}? : \mathbb{P}\ \textit{FEDERATE} \\
\textit{obj}? : \textit{Object} \\
\textit{cattrs}? : \mathbb{P}\ \textit{CLASSATTR} \\
\textit{oattrs} : \mathbb{P}\ \textit{OBJECTATTR} \\
\hline
\textit{ObjectAttrToClassAtttr}(\!|\textit{oattrs}|\!) = \textit{cattrs}? \\
\textit{ObjectAttrToObject}(\!|\textit{oattrs}|\!) = \{\textit{obj}?\} \\
\textit{fed}? \in \textit{Federates} \\
\{\textit{fed}?\} \times \textit{oattrs} \subseteq \textit{Owns} \\
\\
\textit{WillingToDivest}' = \textit{WillingToDivest} \cup (\{\textit{fed}?\} \times \textit{oattrs}) \\
\textit{WillingToAccept}' = \textit{WillingToAccept} \\
\textit{TargetOwners}' = \textit{TargetOwners} \cup (\textit{targets}? \times \textit{oattrs}) \\
\\
\textit{SimulationState}' = \textit{SimulationState} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{RequestAttrOwnAssumption} \\
\Xi \textit{ExecutionState} \\
\textit{fed}? : \textit{FEDERATE} \\
\textit{obj}? : \textit{Object} \\
\textit{cattrs}? : \mathbb{P}\ \textit{CLASSATTR} \\
\textit{oattrs} : \mathbb{P}\ \textit{OBJECTATTR} \\
\hline
\textit{ObjectAttrToClassAtttr}(\textit{oattrs}) = \textit{cattrs}? \\
\textit{ObjectAttrToObject}(\textit{oattrs}) = \{\textit{obj}?\} \\
\textit{fed}? \in \textit{Federates} \\
(\{\textit{fed}?\} \times \textit{oattrs}) \mathbin{;} \textit{ObjectAttrToClassAttr} \subseteq \textit{Publishing} \\
(\{\textit{fed}?\} \times \textit{oattrs}) \cap \textit{Owns} = \varnothing \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{RequestAttrOwnAcquisition} \\
\Delta \textit{ExecutionState} \\
\textit{fed}? : \textit{FEDERATE} \\
\textit{obj}? : \textit{Object} \\
\textit{cattrs}? : \mathbb{P}\ \textit{CLASSATTR} \\
\textit{oattrs} : \mathbb{P}\ \textit{OBJECTATTR} \\
\hline
\textit{ObjectAttrToClassAtttr}(\textit{oattrs}) = \textit{cattrs}? \\
\textit{ObjectAttrToObject}(\textit{oattrs}) = \{\textit{obj}?\} \\
\textit{fed}? \in \textit{Federates} \\
(\{\textit{fed}?\} \times \textit{oattrs}) \mathbin{;} \textit{ObjectAttrToClassAttr} \subseteq \textit{Publishing} \\
(\{\textit{fed}?\} \times \textit{oattrs}) \cap \textit{Owns} = \varnothing \\
\\
\textit{SimulationState}' = \textit{SimulationState} \\
\\
\textit{WillingToAccept}' = \textit{WillingToAccept} \cup (\{\textit{fed}?\} \times \textit{oattrs}) \\
\textit{WillingToDivest}' = \textit{WillingToDivest} \\
\textit{TargetOwners}' = \textit{TargetOwners} \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textbf{AttrOwnDivestNotify} \\
\hline
\Delta ExecutionState \\
fed? : FEDERATE \\
obj? : Object \\
cattrs? : \mathbb{P}\ CLASSATTR \\
oattrs : \mathbb{P}\ OBJECTATTR \\
\hline
ObjectAttrToClassAtttr(oattrs) = cattrs? \\
ObjectAttrToObject(oattrs) = \{obj?\} \\
fed? \in Federates \\
\{fed?\} \times oattrs \subseteq Owns \\
Owns' = Owns \setminus (\{fed?\} \times oattrs) \\
Objects' = Objects \\
Publishing' = Publishing \\
ObjectAttrs' = ObjectAttrs \\
Federates' = Federates \\
WillingToAccept' = WillingToAccept \\
WillingToDivest' = WillingToDivest \setminus (\{fed?\} \times oattrs) \\
TargetOwners' = TargetOwners \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textbf{AttrOwnAcquisitionNotify} \\
\hline
\Delta ExecutionState \\
fed? : FEDERATE \\
obj? : Object \\
cattrs? : \mathbb{P}\ CLASSATTR \\
oattrs : \mathbb{P}\ OBJECTATTR \\
\hline
ObjectAttrToClassAtttr(oattrs) = cattrs? \\
ObjectAttrToObject(oattrs) = \{obj?\} \\
fed? \in Federates \\
Owns^{\sim}(oattrs) = \emptyset \\
oattrs \subseteq TargetOwners(\{fed?\}) \\
Owns' = Owns \cup (\{fed?\} \times oattrs) \\
Objects' = Objects \\
Publishing' = Publishing \\
ObjectAttrs' = ObjectAttrs \\
Federates' = Federates \\
WillingToAccept' = WillingToAccept \setminus (\{fed?\} \times oattrs) \\
WillingToDivest' = WillingToDivest \\
TargetOwners' = TargetOwners \rhd oattrs \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textbf{RequestAttrOwnRelease} \\
\hline
\Xi \mathit{ExecutionState} \\
\mathit{fed}? : \mathit{FEDERATE} \\
\mathit{obj}? : \mathit{Object} \\
\mathit{cattrs}? : \mathbb{P}\ \mathit{CLASSATTR} \\
\mathit{oattrs} : \mathbb{P}\ \mathit{OBJECTATTR} \\
\hline
\mathit{ObjectAttrToClassAtttr}(\mathit{oattrs}) = \mathit{cattrs}? \\
\mathit{ObjectAttrToObject}(\mathit{oattrs}) = \{\mathit{obj}?\} \\
\mathit{fed}? \in \mathit{Federates} \\
(\ \{\mathit{fed}?\} \times \mathit{oattrs}) = \mathit{Owns}
\end{array}
$$

$$
\begin{array}{l}
\textbf{PublishObjectClass} \\
\hline
\Delta \mathit{ExecutionState} \\
\mathit{fed}? : \mathit{FEDERATE} \\
\mathit{class}? : \mathit{CLASS} \\
\mathit{cattrs}? : \mathbb{P}\ \mathit{CLASSATTR} \\
\hline
\mathit{AttributesToClass}(\mathit{cattrs}?) = \{\mathit{class}?\} \land \mathit{cattrs}? = \emptyset \\
\mathit{fed}? \in \mathit{Federates} \\
\mathit{Publishing}' = \mathit{Publishing} \setminus (\ \{\mathit{fed}?\} \lhd \mathit{Publishing} \rhd \mathit{AttributesToClass}\sim(\{\mathit{class}?\})) \\
\qquad\qquad\qquad \cup (\{\mathit{fed}?\} \times \mathit{cattrs}?) \\
\mathit{Owns}' = \mathit{Owns} \setminus (\{\mathit{fed}?\} \lhd \mathit{Owns} \rhd (\mathit{ObjectAttrToClassAttr} \mathbin{;} \mathit{AttributesToClass})\sim(\{\mathit{class}?\},) \\
\mathit{ObjectCollection}' = \mathit{ObjectCollection} \\
\mathit{OwnershipInternalState}' = \mathit{OwnershipInternalState}
\end{array}
$$

$$
\begin{array}{l}
\textbf{UnpublishObjectClass} \\
\hline
\Delta \mathit{ExecutionState} \\
\mathit{fed}? : \mathit{FEDERATE} \\
\mathit{class}? : \mathit{CLASS} \\
\hline
\mathit{class}? \in \mathit{AttributesToClass}(\mathit{Publishing}(\{\mathit{fed}?\})\ ) \\
\mathit{fed}? \in \mathit{Federates} \\
\mathit{Publishing}' = \mathit{Publishing} \setminus (\ \{\mathit{fed}?\} \lhd \mathit{Publishing} \rhd \mathit{AttributesToClass}\sim(\{\mathit{class}?\})) \\
\mathit{Owns}' = \mathit{Owns} \setminus (\{\mathit{fed}?\} \lhd \mathit{Owns} \rhd (\mathit{ObjectAttrToClassAttr} \mathbin{;} \mathit{AttributesToClass})\sim(\{\mathit{class}?\},) \\
\mathit{ObjectCollection}' = \mathit{ObjectCollection} \\
\mathit{OwnershipInternalState}' = \mathit{OwnershipInternalState}
\end{array}
$$

Appendix B. Summary of Selected Z Operators Used

Name	**Operator**	**Definition**
Powerset	$\mathbb{P}\ set$	$\{ s \mid s \subseteq set \}$
Cross product	$set1 \times set2$	$\{ (x,y) \mid x \in set1 \wedge y \in set2 \}$
Relations	$set1 \leftrightarrow set2$	$\mathbb{P}\ (set1 \times set2)$
Partial functions	$set1 \rightarrow\!\!\!\!\!+\ set2$	$\{ f : set1 \leftrightarrow set2 \mid ((x,y) \in f \wedge (x,z) \in f) \Rightarrow y = z \}$
Total functions	$set1 \longrightarrow set2$	$\{ f : set1 \rightarrow\!\!\!\!\!+\ set2 \mid \mathrm{dom}\ f = set1 \}$
Total surjections	$set1 \twoheadrightarrow set2$	$\{ f : set1 \longrightarrow set2 \mid \mathrm{ran}\ f = set2 \}$
Domain	$\mathrm{dom}\ rel$	$\{ x \mid \exists y.(x,y) \in rel \}$
Range	$\mathrm{ran}\ rel$	$\{ y \mid \exists x.(x,y) \in rel \}$
Domain restriction	$set \lhd rel$	$\{ (x,y) \mid (x,y) \in rel \wedge x \in set \}$
Range restriction	$rel \rhd set$	$\{ (x,y) \mid (x,y) \in rel \wedge y \in set \}$
Range anti-restriction	$rel \ntriangleright set$	$\{ (x,y) \mid (x,y) \in rel \wedge y \notin set \}$
Relational inverse	$rel \sim$	$\{ (y,x) \mid (x,y) \in rel)$
Relational composition	$rel1\ ;\ rel2$	$\{ (x,z) \mid \exists y.\ (\ (x,y) \in rel1 \wedge (y,z) \in rel2\ )\ \}$
Relational image	$rel(\!|set|\!)$	$\{ y \mid \exists y.\ (\ (x,y) \in rel \wedge y \in set\ )\ \}$
Relational difference	$rel1 \setminus rel2$	$\{ (x,y) \mid (x,y) \in rel1 \wedge (x,y) \notin rel2\ )\ \}$

Appendix C. NP Description of the HLA Ownership Properties.

```
/* Define the basic kinds of entities to consider */
[CLASS, ATTR, FED, OATTR, OBJECT]

/* Define the basic universe */
ObjectCollection = [
  Objects: set OBJECT
  Object_Attrs: set OATTR
  ObjectToClass: tot OBJECT -> CLASS
  ClassAttrsToClass: tot ATTR -> CLASS
  ObjAttrsToClassAttrs: tot OATTR -> ATTR
  ObjAttrsToObject: tot OATTR -> OBJECT
|
   /* Only object attributes about known objects are of interest */
   Object_Attrs = dom (ObjAttrsToObject :> Objects)
]


GoodObjColl = [
  ObjectCollection
|
  ObjectToClass;ClassAttrsToClass~ = ObjAttrsToObject~;ObjAttrsToClassAttrs
  (ObjAttrsToObject;ObjAttrsToObject~ &
      ObjAttrsToClassAttrs;ObjAttrsToClassAttrs~) <= Id

/*  First invariant says that each instance has the attributes
      specified by its class (or has the right number of attributes

      2nd invariant states that the intersection of the
      two equivalence relations on AttrTo Object and ObjAttrsTo
      ClassAttributes intersect only when the same object attributes
      are the subject, i.e., two object attributes can't be of the
      same type and belong to the same object instance  */

]


/* Explicitly defined state */
SimState = [
  GoodObjColl
  Federates: set FED
  Publishing: FED <-> ATTR
  Owns: FED <-> OATTR
]

/* Implicitly defined state */
OwnershipInternalState = [
  WillingToDivest:FED <-> OATTR
  WillingToAccept: FED <-> OATTR
  TargetOwners : FED <-> OATTR
]

/* Total state to consider */
ExecutionState =
[SimState
OwnershipInternalState]
```

```
/* Define any properties of the state */

 NoTwoOwners = [ SimState | fun Owns~ ]

/* Force a non-empty state */
NonEmpty =  [
  SimState
|
  Publishing  != {}
  Owns != {}
  Federates != {}
]

/* Check that the non-empty state allows two owners */
NoTwoOwnersForced = [NonEmpty | fun Owns~]

NoBadOwnedAttrs  =  [SimState | ran Owns <= Object_Attrs ]

NoBadOwners = [SimState | dom Owns <= Federates]

OwnsOnlyIfPublishes = [SimState | Owns;ObjAttrsToClassAttrs <= Publishing ]

SoundOwners = [
  NoTwoOwners
  NoBadOwnedAttrs
  NoBadOwners
  OwnsOnlyIfPublishes
]

CompleteOwners  = [SimState | ran Owns = Object_Attrs]

/* Properties defined about implicit state */
SoundDivestments = [ExecutionState | WillingToDivest <= Owns]

SoundAccepts = [ExecutionState | WillingToAccept & Owns = {} ]

TargetsUnowned = [ExecutionState | ran TargetOwners & ran Owns = {} ]
```

```
/* Operations defined on the state */

CreateFedExecution() =
[
  ExecutionState
  SimState
|
  Objects' = {}
  Object_Attrs' = {}
  Federates' = {}
  Publishing' = {}
  Owns' = {}
  WillingToAccept' = {}
  WillingToDivest' = {}
  TargetOwners' = {}
]


JoinFedExecution(fed?:FED) =
[
  ExecutionState
  const ObjectCollection
  const OwnershipInternalState
|
  fed? not in Federates

  Federates' = (Federates U {fed?})
  Publishing = Publishing'
  Owns' = Owns
]


RequestAttrOwnDivestiture(fed?:FED, obj?:OBJECT, targets?:set FED,
                              oattrs?:set OATTR) =
[
  ExecutionState
  const SimState
|
  fed? in Federates
  ObjAttrsToObject.oattrs? = {obj?}
  obj? in Objects
  ({fed?} <: Un :> oattrs?) <= Owns

  WillingToDivest' = WillingToDivest U ({fed?} <: Un :> oattrs?)
  WillingToAccept' = WillingToAccept
  TargetOwners' = TargetOwners U (targets? <: Un :> oattrs?)
]
```

```
RequestAttrOwnAssumption(fed?:FED, obj?:OBJECT, oattrs?:set OATTR) =
[
  const ExecutionState
|
  fed? in Federates
  ObjAttrsToObject.oattrs? = {obj?}
  obj? in Objects
  ({fed?} <: Un :> oattrs?);ObjAttrsToClassAttrs <= Publishing
  ({fed?} <: Un :> oattrs?) & Owns = {}
]


RequestAttrOwnAcquisition(fed?:FED, obj?:OBJECT, oattrs?:set OATTR) =
[
  ExecutionState
  const SimState
|
  fed? in Federates
  ObjAttrsToObject.oattrs? = {obj?}
  obj? in Objects
  ({fed?} <: Un :> oattrs?);ObjAttrsToClassAttrs <= Publishing
  ({fed?} <: Un :> oattrs?) & Owns = {}
  WillingToDivest' = WillingToDivest
  WillingToAccept' = WillingToAccept U ({fed?} <: Un :> oattrs?)
  TargetOwners' = TargetOwners
]


AttrOwnDivestNotify(fed?:FED, obj?:OBJECT, oattrs?:set OATTR) =
[
  ExecutionState
  const ObjectCollection
|
  fed? in Federates
  ObjAttrsToObject.oattrs? = {obj?}
  obj? in Objects
  ({fed?} <: Un :> oattrs?) <= Owns
  ({fed?} <: Un :> oattrs?) <= WillingToDivest
  Owns' = Owns \ ({fed?} <: Un :> oattrs?)
  Federates' = Federates
  Publishing' = Publishing
  WillingToDivest' = WillingToDivest \ ({fed?} <: Un :> oattrs?)
  WillingToAccept' = WillingToAccept
  TargetOwners' = TargetOwners
]
```

```
AttrOwnAcquisitionNotify(fed?:FED, obj?:OBJECT, oattrs?:set OATTR) =
[
  ExecutionState
  const ObjectCollection
|
  fed? in Federates
  ObjAttrsToObject.oattrs? = {obj?}
  Owns~.oattrs? = {}
  /* Only look for owners amongst the target owners */
  obj? in Objects
  oattrs? <= TargetOwners.{fed?}
  ({fed?} <: Un :> oattrs?) <= WillingToAccept

  Owns' = Owns U ({fed?} <: Un :> oattrs?)
  Federates' = Federates
  Publishing' = Publishing

  WillingToAccept' = WillingToAccept \ ({fed?} <: Un :> oattrs?)
  WillingToDivest' = WillingToDivest
  TargetOwners' = TargetOwners ;> oattrs?
]
```

```
RequestAttrOwnRelease(fed?:FED, obj?:OBJECT, oattrs?:set OATTR) =
[
  const ExecutionState
|
  fed? in Federates
  ObjAttrsToObject.oattrs? = {obj?}
  obj? in Objects
  ({fed?} <: Un :> oattrs?) <= Owns
]
```

```
PublishObjectClass(fed?:FED, class?:CLASS, cattrs?: set ATTR) =
[
  ExecutionState
  const ObjectCollection
  const OwnershipInternalState
|
  ClassAttrsToClass.cattrs? = {class?} or cattrs? = {}
  fed? in Federates

  Federates' = Federates
  Publishing' = Publishing \ ( {fed?} <: Publishing :>
                                        (ClassAttrsToClass~.{class?}))
                                    U ({fed?} <: Un :> cattrs?)
  Owns' = Owns \ ({fed?} <: Owns :>
                   ((ObjAttrsToClassAttrs;ClassAttrsToClass)~.{class?}))

]
```

```
UnpublishObjectClass(fed?:FED, class?:CLASS) =
[
  ExecutionState
  const ObjectCollection
  const OwnershipInternalState
|
  fed? in Federates
  class? in ClassAttrsToClass.(Publishing.{fed?})

  Federates' = Federates
  Publishing' = Publishing \ ( {fed?} <: Publishing :>
                                    (ClassAttrsToClass~.{class?}))
  Owns' = Owns \ ({fed?} <: Owns :>
                         ((ObjAttrsToClassAttrs;ClassAttrsToClass)~.{class?}))
]
```

```
/* Now construct the claims to test */

/* Check that each modifying operation maintains sound ownership */
ReqAttrDivSoundOwns(fed:FED, obj:OBJECT, targets:set FED, oattrs:set OATTR)::
   SoundOwners and RequestAttrOwnDivestiture(fed,obj,targets,oattrs) =>
SoundOwners'

ReqAttrAcqSoundOwns(fed:FED, obj:OBJECT, oattrs:set OATTR)::
   SoundOwners and RequestAttrOwnAcquisition(fed,obj,oattrs) => SoundOwners'

AttrDivNotSoundOwns(fed:FED, obj:OBJECT, oattrs:set OATTR)::
   SoundOwners and AttrOwnDivestNotify(fed,obj,oattrs) => SoundOwners'

AttrAcqNotSoundOwns(fed:FED, obj:OBJECT, oattrs:set OATTR)::
   SoundOwners and AttrOwnAcquisitionNotify(fed,obj,oattrs) => SoundOwners'

PublishSoundOwns(fed:FED, class:CLASS, cattrs:set ATTR)::
   SoundOwners and PublishObjectClass(fed,class,cattrs) => SoundOwners'

UnpublishSoundOwns(fed:FED, class:CLASS)::
   SoundOwners and UnpublishObjectClass(fed,class) => SoundOwners'

/* Check that willing to divest and accept stays sound */

ReqAttrDivSoundDiv(fed:FED, obj:OBJECT, targets:set FED, oattrs:set OATTR)::
   SoundDivestments and RequestAttrOwnDivestiture(fed,obj,targets,oattrs) =>
SoundDivestments'

AttrDivNotSoundDiv(fed:FED, obj:OBJECT, oattrs:set OATTR)::
   SoundDivestments and AttrOwnDivestNotify(fed,obj,oattrs) =>
SoundDivestments'

ReqAttrAcqSoundAcc(fed:FED, obj:OBJECT, oattrs:set OATTR)::
   SoundAccepts and RequestAttrOwnAcquisition(fed,obj,oattrs) => SoundAccepts'

AttrAcqNotSoundAcc(fed:FED, obj:OBJECT, oattrs:set OATTR)::
   SoundAccepts and AttrOwnAcquisitionNotify(fed,obj,oattrs) => SoundAccepts'
```

```
/*************************************************************************/
/* Check against protocol executions, not just single operations         */
/*************************************************************************/

/* Check for complete ownership after a simple conditional divestiture */

ConditionalCompleteOwners(fed1:FED, fed2:FED, targets : set FED, obj:OBJECT,
                                     oattrs1:set OATTR, oattrs2:set OATTR)::
[
  ExecutionState
|
  /* require the case we are interested in */
  not fed2 = fed1 and
  fed2 in targets and
  oattrs2 <= oattrs1 and
  SoundOwners and

  CompleteOwners and

  /* the conditional divestiture of oattrs1, actually divesting oattrs2 */

     (RequestAttrOwnDivestiture(fed1,obj,targets,oattrs1);
      RequestAttrOwnAssumption(fed2,obj,oattrs1);
      RequestAttrOwnAcquisition(fed2,obj,oattrs2);
      AttrOwnDivestNotify(fed1,obj,oattrs2);
      AttrOwnAcquisitionNotify(fed2,obj,oattrs2)) =>
  CompleteOwners'
]


/* How about an unpublish in the middle of an acquisition */

UnpublishInAcquisition(fed:FED, obj:OBJECT, oattr : OATTR)::
[
  ObjectCollection
  const class : CLASS
|
  class = ClassAttrsToClass.(ObjAttrsToClassAttrs.oattr)
  obj = ObjAttrsToObject.oattr

  SoundOwners and
     RequestAttrOwnAcquisition(fed,obj,{oattr});
     UnpublishObjectClass(fed,class);
     AttrOwnAcquisitionNotify(fed,obj,{oattr})  =>
  SoundOwners'
]
```

```
/* Now check that target owners is maintained correctly when ownership is
transferred unconditionally */
UnconditionalSoundTargets(fed1:FED, obj:OBJECT, targets:set FED,
                        oattrs1:set OATTR, fed2:FED, oattrs2:set OATTR)::
[
|
  /* require the case we are interested in */
  not fed2 = fed1 and
  oattrs2 <= oattrs1 and
  SoundOwners and
  TargetsUnowned and
     (RequestAttrOwnDivestiture(fed1,obj,targets,oattrs1);
      AttrOwnDivestNotify(fed1,obj,oattrs1);
      AttrOwnAcquisitionNotify(fed2,obj,oattrs2)) =>
  TargetsUnowned'
]


/* And check for conditionally as well */
ConditionalSoundTargets(fed1:FED, obj:OBJECT, targets:set FED,
                        oattrs1:set OATTR,fed2:FED, oattrs2:set OATTR)::
[
  ExecutionState
|
  /* require the case we are interested in */
  not fed2 = fed1 and
  oattrs2 <= oattrs1 and
  SoundOwners and
  /* the targetowners still owned should be owned by the originating and be
willing to divest */
  TargetsUnowned and
     (RequestAttrOwnDivestiture(fed1,obj,targets,oattrs1);
      AttrOwnDivestNotify(fed1,obj,oattrs2);
      AttrOwnAcquisitionNotify(fed2,obj,oattrs2)) =>
   (ran TargetOwners' & ran Owns' = oattrs1 \ oattrs2 and
    dom (Owns' :> (oattrs1 \ oattrs2)) <= { fed1 } and
    {fed1} <: Un :> (oattrs1 \ oattrs2) <= WillingToDivest')
]
```

Appendix D. Ladybug results.

Log run by script hla-own-tr.script2 at Wed Apr 14 23:16:50 EDT 1999
checking hla ownership properties
11:16:51 PM : Parsing tests/hla4.np...
11:16:56 PM : tests/hla4.np loaded successfully

****
now check chosen scope
****

check that Request Attribute Divestiture maintains sound ownership properties

#FED = 2
#OBJECT = 3
#ATTR = 2
#OATTR = 6
#CLASS = 1
11:16:57 PM : Computing facts for ReqAttrDivSoundOwns clause 1 ...
11:16:58 PM : Translating ReqAttrDivSoundOwns clause 1 ...
Completed translation of ReqAttrDivSoundOwns clause 1
Required 0:00:04.7 starting at 11:16:57 PM
11:17:02 PM : Checking ReqAttrDivSoundOwns clause 1 ...
Completed checking ReqAttrDivSoundOwns clause 1
Found 0 Counterexamples
Checked 38,973 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 3.555633e50 total values and 2.844367e50 total cases
Required 0:00:00.9 starting at 11:17:02 PM
11:17:03 PM : Checking ReqAttrDivSoundOwns clause 2
11:17:03 PM : Computing facts for ReqAttrDivSoundOwns clause 2 ...
11:17:03 PM : Translating ReqAttrDivSoundOwns clause 2 ...
Completed translation of ReqAttrDivSoundOwns clause 2
Required 0:00:00.9 starting at 11:17:03 PM
11:17:04 PM : Checking ReqAttrDivSoundOwns clause 2 ...
Completed checking ReqAttrDivSoundOwns clause 2
Found 0 Counterexamples
Checked 151,485 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 3.555633e50 total values and 2.844367e50 total cases
Required 0:00:01.6 starting at 11:17:04 PM
11:17:06 PM : Checking ReqAttrDivSoundOwns clause 3
11:17:06 PM : Computing facts for ReqAttrDivSoundOwns clause 3 ...

11:17:06 PM : Translating ReqAttrDivSoundOwns clause 3 ...
Completed translation of ReqAttrDivSoundOwns clause 3
Required 0:00:01.0 starting at 11:17:06 PM
11:17:07 PM : Checking ReqAttrDivSoundOwns clause 3 ...
Completed checking ReqAttrDivSoundOwns clause 3
Found 0 Counterexamples
Checked 38,955 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 3.555633e50 total values and 2.844367e50 total cases
Required 0:00:00.4 starting at 11:17:07 PM
11:17:07 PM : Checking ReqAttrDivSoundOwns clause 4
11:17:07 PM : Computing facts for ReqAttrDivSoundOwns clause 4 ...
11:17:07 PM : Translating ReqAttrDivSoundOwns clause 4 ...
Completed translation of ReqAttrDivSoundOwns clause 4
Required 0:00:00.8 starting at 11:17:07 PM
11:17:08 PM : Checking ReqAttrDivSoundOwns clause 4 ...
Completed checking ReqAttrDivSoundOwns clause 4
Found 0 Counterexamples
Checked 332,893 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 3.555633e50 total values and 2.844367e50 total cases
Required 0:00:04.1 starting at 11:17:08 PM
11:17:12 PM : Completed checking ReqAttrDivSoundOwns
Completed checking ReqAttrDivSoundOwns
Found 0 Counterexamples
Checked 0 cases and 562,306 values
Assignment space includes 1.422253e51 total values and 1.137747e51 total cases
Required 0:00:07.1 starting at 11:16:57 PM
11:17:12 PM : Completed checking Claim ReqAttrDivSoundOwns

check that Request Attribute Acquisition maintains sound ownership properties

#FED = 2
#OBJECT = 3
#ATTR = 2
#OATTR = 6
#CLASS = 1
11:17:12 PM : Computing facts for ReqAttrAcqSoundOwns clause 1 ...
11:17:12 PM : Translating ReqAttrAcqSoundOwns clause 1 ...
Completed translation of ReqAttrAcqSoundOwns clause 1
Required 0:00:00.3 starting at 11:17:12 PM
11:17:13 PM : Checking ReqAttrAcqSoundOwns clause 1 ...

```
Completed checking ReqAttrAcqSoundOwns clause 1
Found 0 Counterexamples
Checked 75,297 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 7.112655e49 total values and 7.110918e49 total cases
Required 0:00:00.8 starting at 11:17:13 PM
11:17:13 PM : Checking ReqAttrAcqSoundOwns clause 2
11:17:13 PM : Computing facts for ReqAttrAcqSoundOwns clause 2 ...
11:17:13 PM : Translating ReqAttrAcqSoundOwns clause 2 ...
Completed translation of ReqAttrAcqSoundOwns clause 2
Required 0:00:00.6 starting at 11:17:13 PM
11:17:14 PM : Checking ReqAttrAcqSoundOwns clause 2 ...
Completed checking ReqAttrAcqSoundOwns clause 2
Found 0 Counterexamples
Checked 296,781 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 7.112655e49 total values and 7.110918e49 total cases
Required 0:00:02.9 starting at 11:17:14 PM
11:17:17 PM : Checking ReqAttrAcqSoundOwns clause 3
11:17:17 PM : Computing facts for ReqAttrAcqSoundOwns clause 3 ...
11:17:17 PM : Translating ReqAttrAcqSoundOwns clause 3 ...
Completed translation of ReqAttrAcqSoundOwns clause 3
Required 0:00:00.6 starting at 11:17:17 PM
11:17:17 PM : Checking ReqAttrAcqSoundOwns clause 3 ...
Completed checking ReqAttrAcqSoundOwns clause 3
Found 0 Counterexamples
Checked 75,297 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 7.112655e49 total values and 7.110918e49 total cases
Required 0:00:00.8 starting at 11:17:17 PM
11:17:18 PM : Checking ReqAttrAcqSoundOwns clause 4
11:17:18 PM : Computing facts for ReqAttrAcqSoundOwns clause 4 ...
11:17:18 PM : Translating ReqAttrAcqSoundOwns clause 4 ...
Completed translation of ReqAttrAcqSoundOwns clause 4
Required 0:00:00.7 starting at 11:17:18 PM
11:17:19 PM : Checking ReqAttrAcqSoundOwns clause 4 ...
Completed checking ReqAttrAcqSoundOwns clause 4
Found 0 Counterexamples
Checked 1,431,621 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 7.112655e49 total values and 7.110918e49 total cases
Required 0:00:15.4 starting at 11:17:19 PM
11:17:34 PM : Completed checking ReqAttrAcqSoundOwns
```

Completed checking ReqAttrAcqSoundOwns
Found 0 Counterexamples
Checked 0 cases and 1,878,996 values
Assignment space includes 2.845062e50 total values and 2.844367e50 total cases
Required 0:00:19.8 starting at 11:17:12 PM
11:17:34 PM : Completed checking Claim ReqAttrAcqSoundOwns

check that Attribute Divestiture Notification maintains sound ownership properties

#FED = 2
#OBJECT = 3
#ATTR = 2
#OATTR = 6
#CLASS = 1
11:17:34 PM : Computing facts for AttrDivNotSoundOwns clause 1 ...
11:17:34 PM : Translating AttrDivNotSoundOwns clause 1 ...
Completed translation of AttrDivNotSoundOwns clause 1
Required 0:00:00.4 starting at 11:17:34 PM
11:17:35 PM : Checking AttrDivNotSoundOwns clause 1 ...
Completed checking AttrDivNotSoundOwns clause 1
Found 0 Counterexamples
Checked 34,505 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 7.112655e49 total values and 7.110918e49 total cases
Required 0:00:00.5 starting at 11:17:35 PM
11:17:35 PM : Checking AttrDivNotSoundOwns clause 2
11:17:35 PM : Computing facts for AttrDivNotSoundOwns clause 2 ...
11:17:35 PM : Translating AttrDivNotSoundOwns clause 2 ...
Completed translation of AttrDivNotSoundOwns clause 2
Required 0:00:00.6 starting at 11:17:35 PM
11:17:36 PM : Checking AttrDivNotSoundOwns clause 2 ...
Completed checking AttrDivNotSoundOwns clause 2
Found 0 Counterexamples
Checked 53,873 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 7.112655e49 total values and 7.110918e49 total cases
Required 0:00:00.6 starting at 11:17:36 PM
11:17:36 PM : Checking AttrDivNotSoundOwns clause 3
11:17:36 PM : Computing facts for AttrDivNotSoundOwns clause 3 ...
11:17:37 PM : Translating AttrDivNotSoundOwns clause 3 ...
Completed translation of AttrDivNotSoundOwns clause 3
Required 0:00:00.6 starting at 11:17:36 PM

11:17:37 PM : Checking AttrDivNotSoundOwns clause 3 ...
Completed checking AttrDivNotSoundOwns clause 3
Found 0 Counterexamples
Checked 34,505 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 7.112655e49 total values and 7.110918e49 total cases
Required 0:00:00.4 starting at 11:17:37 PM
11:17:37 PM : Checking AttrDivNotSoundOwns clause 4
11:17:37 PM : Computing facts for AttrDivNotSoundOwns clause 4 ...
11:17:38 PM : Translating AttrDivNotSoundOwns clause 4 ...
Completed translation of AttrDivNotSoundOwns clause 4
Required 0:00:00.9 starting at 11:17:37 PM
11:17:38 PM : Checking AttrDivNotSoundOwns clause 4 ...
Completed checking AttrDivNotSoundOwns clause 4
Found 0 Counterexamples
Checked 182,835 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 7.112655e49 total values and 7.110918e49 total cases
Required 0:00:01.7 starting at 11:17:38 PM
11:17:40 PM : Completed checking AttrDivNotSoundOwns
Completed checking AttrDivNotSoundOwns
Found 0 Counterexamples
Checked 0 cases and 305,718 values
Assignment space includes 2.845062e50 total values and 2.844367e50 total cases
Required 0:00:03.2 starting at 11:17:34 PM
11:17:40 PM : Completed checking Claim AttrDivNotSoundOwns

check that Attribute Acquisition Notification maintains sound ownership properties

#FED = 2
#OBJECT = 3
#ATTR = 2
#OATTR = 6
#CLASS = 1
11:17:40 PM : Computing facts for AttrAcqNotSoundOwns clause 1 ...
11:17:40 PM : Translating AttrAcqNotSoundOwns clause 1 ...
Completed translation of AttrAcqNotSoundOwns clause 1
Required 0:00:00.4 starting at 11:17:40 PM
11:17:41 PM : Checking AttrAcqNotSoundOwns clause 1 ...
Completed checking AttrAcqNotSoundOwns clause 1
Found 0 Counterexamples
Checked 23,919 values and 0 cases

```
Covered 100% of the total assignment space
Assignment space includes 7.112655e49 total values and 7.110918e49 total cases
Required 0:00:00.4 starting at 11:17:41 PM
11:17:41 PM : Checking AttrAcqNotSoundOwns clause 2
11:17:41 PM : Computing facts for AttrAcqNotSoundOwns clause 2 ...
11:17:41 PM : Translating AttrAcqNotSoundOwns clause 2 ...
Completed translation of AttrAcqNotSoundOwns clause 2
Required 0:00:00.6 starting at 11:17:41 PM
11:17:42 PM : Checking AttrAcqNotSoundOwns clause 2 ...
Completed checking AttrAcqNotSoundOwns clause 2
Found 0 Counterexamples
Checked 23,919 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 7.112655e49 total values and 7.110918e49 total cases
Required 0:00:00.3 starting at 11:17:42 PM
11:17:42 PM : Checking AttrAcqNotSoundOwns clause 3
11:17:42 PM : Computing facts for AttrAcqNotSoundOwns clause 3 ...
11:17:42 PM : Translating AttrAcqNotSoundOwns clause 3 ...
Completed translation of AttrAcqNotSoundOwns clause 3
Required 0:00:00.7 starting at 11:17:42 PM
11:17:43 PM : Checking AttrAcqNotSoundOwns clause 3 ...
Completed checking AttrAcqNotSoundOwns clause 3
Found 0 Counterexamples
Checked 23,919 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 7.112655e49 total values and 7.110918e49 total cases
Required 0:00:00.3 starting at 11:17:43 PM
11:17:43 PM : Checking AttrAcqNotSoundOwns clause 4
11:17:43 PM : Computing facts for AttrAcqNotSoundOwns clause 4 ...
11:17:43 PM : Translating AttrAcqNotSoundOwns clause 4 ...
Completed translation of AttrAcqNotSoundOwns clause 4
Required 0:00:01.0 starting at 11:17:43 PM
11:17:44 PM : Checking AttrAcqNotSoundOwns clause 4 ...

Found Counterexample to Claim AttrAcqNotSoundOwns:
ClassAttrsToClass : tot ATTR->CLASS =
  { a0 -> c0,
 a1 -> c0 }
ClassAttrsToClass' : tot ATTR->CLASS =
  { a0 -> c0,
 a1 -> c0 }
fed : FED =
  f0
```

```
Federates : set FED =
  { f0 }
Federates' : set FED =
  { f0 }
oattrs : set OATTR =
  { oa0 }
obj : OBJECT =
  ob0
ObjAttrsToClassAttrs : tot OATTR->ATTR =
  { oa0 -> a0,
 oa1 -> a1,
 oa2 -> a0,
 oa3 -> a1,
 oa4 -> a0,
 oa5 -> a1 }
ObjAttrsToClassAttrs' : tot OATTR->ATTR =
  { oa0 -> a0,
 oa1 -> a1,
 oa2 -> a0,
 oa3 -> a1,
 oa4 -> a0,
 oa5 -> a1 }
ObjAttrsToObject : tot OATTR->OBJECT =
  { oa0 -> ob0,
 oa1 -> ob0,
 oa2 -> ob1,
 oa3 -> ob1,
 oa4 -> ob2,
 oa5 -> ob2 }
ObjAttrsToObject' : tot OATTR->OBJECT =
  { oa0 -> ob0,
 oa1 -> ob0,
 oa2 -> ob1,
 oa3 -> ob1,
 oa4 -> ob2,
 oa5 -> ob2 }
Object_Attrs : set OATTR =
  { oa0, oa1 }
Object_Attrs' : set OATTR =
  { oa0, oa1 }
Objects : set OBJECT =
  { ob0 }
Objects' : set OBJECT =
```

```
  { ob0 }
ObjectToClass : tot OBJECT->CLASS =
  { ob0 -> c0,
 ob1 -> c0,
 ob2 -> c0 }
ObjectToClass' : tot OBJECT->CLASS =
  { ob0 -> c0,
 ob1 -> c0,
 ob2 -> c0 }
Owns : FED<->OATTR =
  {  }
Owns' : FED<->OATTR =
  { f0 -> {oa0 } }
Publishing : FED<->ATTR =
  {  }
Publishing' : FED<->ATTR =
  {  }
TargetOwners : FED<->OATTR =
  { f0 -> {oa0 } }
TargetOwners' : FED<->OATTR =
  {  }
WillingToAccept : FED<->OATTR =
  { f0 -> {oa0 } }
WillingToAccept' : FED<->OATTR =
  {  }
WillingToDivest : FED<->OATTR =
  {  }
WillingToDivest' : FED<->OATTR =
  {  }

11:17:44 PM : Found 1 Counterexamples
Found 1 Counterexamples
Checked 694 values and 1 cases
Covered 1% of the total assignment space
Assignment space includes 7.112655e49 total values and 7.110918e49 total cases
Required 0:00:01.2 starting at 11:17:40 PM
11:17:44 PM : Stopped checking Claim AttrAcqNotSoundOwns
Completed checking AttrAcqNotSoundOwns clause 4
Found 1 Counterexamples
Checked 694 values and 1 cases
Covered 1% of the total assignment space
Assignment space includes 7.112655e49 total values and 7.110918e49 total cases
Required 0:00:00.1 starting at 11:17:44 PM
```

11:17:44 PM : Completed checking AttrAcqNotSoundOwns
Completed checking AttrAcqNotSoundOwns
Found 1 Counterexamples
Checked 1 cases and 72,451 values
Assignment space includes 2.845062e50 total values and 2.844367e50 total cases
Required 0:00:01.2 starting at 11:17:40 PM
11:17:44 PM : Completed checking Claim AttrAcqNotSoundOwns

check that Publishing a class maintains sound ownership properties

#FED = 2
#OBJECT = 3
#ATTR = 2
#OATTR = 6
#CLASS = 1
11:17:44 PM : Computing facts for PublishSoundOwns clause 1 ...
11:17:44 PM : Translating PublishSoundOwns clause 1 ...
Completed translation of PublishSoundOwns clause 1
Required 0:00:00.7 starting at 11:17:44 PM
11:17:45 PM : Checking PublishSoundOwns clause 1 ...
Completed checking PublishSoundOwns clause 1
Found 0 Counterexamples
Checked 10,943 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 1.481838e48 total values and 1.481441e48 total cases
Required 0:00:00.2 starting at 11:17:45 PM
11:17:45 PM : Checking PublishSoundOwns clause 2
11:17:45 PM : Computing facts for PublishSoundOwns clause 2 ...
11:17:45 PM : Translating PublishSoundOwns clause 2 ...
Completed translation of PublishSoundOwns clause 2
Required 0:00:00.9 starting at 11:17:45 PM
11:17:46 PM : Checking PublishSoundOwns clause 2 ...
Completed checking PublishSoundOwns clause 2
Found 0 Counterexamples
Checked 65,318 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 1.481838e48 total values and 1.481441e48 total cases
Required 0:00:00.8 starting at 11:17:46 PM
11:17:47 PM : Checking PublishSoundOwns clause 3
11:17:47 PM : Computing facts for PublishSoundOwns clause 3 ...
11:17:47 PM : Translating PublishSoundOwns clause 3 ...
Completed translation of PublishSoundOwns clause 3

Required 0:00:01.2 starting at 11:17:47 PM
11:17:48 PM : Checking PublishSoundOwns clause 3 ...
Completed checking PublishSoundOwns clause 3
Found 0 Counterexamples
Checked 10,943 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 1.481838e48 total values and 1.481441e48 total cases
Required 0:00:00.1 starting at 11:17:48 PM
11:17:48 PM : Checking PublishSoundOwns clause 4
11:17:48 PM : Computing facts for PublishSoundOwns clause 4 ...
11:17:49 PM : Translating PublishSoundOwns clause 4 ...
Completed translation of PublishSoundOwns clause 4
Required 0:00:01.4 starting at 11:17:48 PM
11:17:49 PM : Checking PublishSoundOwns clause 4 ...
Completed checking PublishSoundOwns clause 4
Found 0 Counterexamples
Checked 58,043 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 1.481838e48 total values and 1.481441e48 total cases
Required 0:00:00.8 starting at 11:17:49 PM
11:17:50 PM : Checking PublishSoundOwns clause 5
11:17:50 PM : Computing facts for PublishSoundOwns clause 5 ...
11:17:51 PM : Translating PublishSoundOwns clause 5 ...
Completed translation of PublishSoundOwns clause 5
Required 0:00:01.1 starting at 11:17:50 PM
11:17:51 PM : Checking PublishSoundOwns clause 5 ...
Completed checking PublishSoundOwns clause 5
Found 0 Counterexamples
Checked 5,043 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 1.482196e48 total values and 1.481441e48 total cases
Required 0:00:00.1 starting at 11:17:51 PM
11:17:51 PM : Checking PublishSoundOwns clause 6
11:17:51 PM : Computing facts for PublishSoundOwns clause 6 ...
11:17:52 PM : Translating PublishSoundOwns clause 6 ...
Completed translation of PublishSoundOwns clause 6
Required 0:00:01.2 starting at 11:17:51 PM
11:17:53 PM : Checking PublishSoundOwns clause 6 ...
Completed checking PublishSoundOwns clause 6
Found 0 Counterexamples
Checked 22,355 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 1.482196e48 total values and 1.481441e48 total cases

```
Required 0:00:00.3 starting at 11:17:53 PM
11:17:53 PM : Checking PublishSoundOwns clause 7
11:17:53 PM : Computing facts for PublishSoundOwns clause 7 ...
11:17:54 PM : Translating PublishSoundOwns clause 7 ...
Completed translation of PublishSoundOwns clause 7
Required 0:00:01.3 starting at 11:17:53 PM
11:17:54 PM : Checking PublishSoundOwns clause 7 ...
Completed checking PublishSoundOwns clause 7
Found 0 Counterexamples
Checked 6,025 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 1.482196e48 total values and 1.481441e48 total cases
Required 0:00:00.1 starting at 11:17:54 PM
11:17:54 PM : Checking PublishSoundOwns clause 8
11:17:54 PM : Computing facts for PublishSoundOwns clause 8 ...
11:17:55 PM : Translating PublishSoundOwns clause 8 ...
Completed translation of PublishSoundOwns clause 8
Required 0:00:01.4 starting at 11:17:54 PM
11:17:56 PM : Checking PublishSoundOwns clause 8 ...
Completed checking PublishSoundOwns clause 8
Found 0 Counterexamples
Checked 19,572 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 1.482196e48 total values and 1.481441e48 total cases
Required 0:00:00.3 starting at 11:17:56 PM
11:17:56 PM : Completed checking PublishSoundOwns
Completed checking PublishSoundOwns
Found 0 Counterexamples
Checked 0 cases and 198,242 values
Assignment space includes 1.185614e49 total values and 1.185153e49 total cases
Required 0:00:02.6 starting at 11:17:44 PM
11:17:56 PM : Completed checking Claim PublishSoundOwns


check that Unpublishing a class maintains sound ownership properties

#FED = 2
#OBJECT = 3
#ATTR = 2
#OATTR = 6
#CLASS = 1
11:17:56 PM : Computing facts for UnpublishSoundOwns clause 1 ...
11:17:56 PM : Translating UnpublishSoundOwns clause 1 ...
```

Completed translation of UnpublishSoundOwns clause 1
Required 0:00:00.3 starting at 11:17:56 PM
11:17:56 PM : Checking UnpublishSoundOwns clause 1 ...
Completed checking UnpublishSoundOwns clause 1
Found 0 Counterexamples
Checked 46,042 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 3.704599e47 total values and 3.703603e47 total cases
Required 0:00:00.5 starting at 11:17:56 PM
11:17:57 PM : Checking UnpublishSoundOwns clause 2
11:17:57 PM : Computing facts for UnpublishSoundOwns clause 2 ...
11:17:57 PM : Translating UnpublishSoundOwns clause 2 ...
Completed translation of UnpublishSoundOwns clause 2
Required 0:00:00.4 starting at 11:17:57 PM
11:17:57 PM : Checking UnpublishSoundOwns clause 2 ...
Completed checking UnpublishSoundOwns clause 2
Found 0 Counterexamples
Checked 71,935 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 3.704599e47 total values and 3.703603e47 total cases
Required 0:00:00.7 starting at 11:17:57 PM
11:17:58 PM : Checking UnpublishSoundOwns clause 3
11:17:58 PM : Computing facts for UnpublishSoundOwns clause 3 ...
11:17:58 PM : Translating UnpublishSoundOwns clause 3 ...
Completed translation of UnpublishSoundOwns clause 3
Required 0:00:00.4 starting at 11:17:58 PM
11:17:58 PM : Checking UnpublishSoundOwns clause 3 ...
Completed checking UnpublishSoundOwns clause 3
Found 0 Counterexamples
Checked 49,750 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 3.704599e47 total values and 3.703603e47 total cases
Required 0:00:00.5 starting at 11:17:58 PM
11:17:59 PM : Checking UnpublishSoundOwns clause 4
11:17:59 PM : Computing facts for UnpublishSoundOwns clause 4 ...
11:17:59 PM : Translating UnpublishSoundOwns clause 4 ...
Completed translation of UnpublishSoundOwns clause 4
Required 0:00:00.4 starting at 11:17:59 PM
11:17:59 PM : Checking UnpublishSoundOwns clause 4 ...
Completed checking UnpublishSoundOwns clause 4
Found 0 Counterexamples
Checked 46,042 values and 0 cases
Covered 100% of the total assignment space

Assignment space includes 3.704599e47 total values and 3.703603e47 total cases
Required 0:00:00.5 starting at 11:17:59 PM
11:18:00 PM : Completed checking UnpublishSoundOwns
Completed checking UnpublishSoundOwns
Found 0 Counterexamples
Checked 0 cases and 213,769 values
Assignment space includes 1.48184e48 total values and 1.481441e48 total cases
Required 0:00:02.2 starting at 11:17:56 PM
11:18:00 PM : Completed checking Claim UnpublishSoundOwns

check that a simple conditional divestiture maintains ownership

#FED = 2
#OBJECT = 1
#ATTR = 3
#OATTR = 3
#CLASS = 1
11:18:00 PM : Computing facts for ConditionalCompleteOwners ...
11:18:01 PM : Translating ConditionalCompleteOwners ...
Completed translation of ConditionalCompleteOwners
Required 0:00:01.9 starting at 11:18:00 PM
11:18:02 PM : Checking ConditionalCompleteOwners ...
11:18:02 PM : Completed checking ConditionalCompleteOwners
Completed checking ConditionalCompleteOwners
Found 0 Counterexamples
Checked 3,764 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 2.287481e64 total values and 2.251739e64 total cases
Required 0:00:00.1 starting at 11:18:00 PM
11:18:02 PM : Completed checking Claim ConditionalCompleteOwners

check that unconditional divestiture leaves all targets unowned

#FED = 2
#OBJECT = 1
#ATTR = 3
#OATTR = 3
#CLASS = 1
11:18:02 PM : Computing facts for UnconditionalSoundTargets ...
11:18:03 PM : Translating UnconditionalSoundTargets ...
Completed translation of UnconditionalSoundTargets

Required 0:00:01.2 starting at 11:18:02 PM
11:18:04 PM : Checking UnconditionalSoundTargets ...
11:18:04 PM : Completed checking UnconditionalSoundTargets
Completed checking UnconditionalSoundTargets
Found 0 Counterexamples
Checked 5,274 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 1.232860e52 total values and 1.213597e52 total cases
Required 0:00:00.1 starting at 11:18:02 PM
11:18:04 PM : Completed checking Claim UnconditionalSoundTargets

check that conditional divestiture leaves all targets unowned

#FED = 2
#OBJECT = 1
#ATTR = 3
#OATTR = 3
#CLASS = 1
11:18:04 PM : Computing facts for ConditionalSoundTargets clause 1 ...
11:18:04 PM : Translating ConditionalSoundTargets clause 1 ...
Completed translation of ConditionalSoundTargets clause 1
Required 0:00:01.5 starting at 11:18:04 PM
11:18:06 PM : Checking ConditionalSoundTargets clause 1 ...
Completed checking ConditionalSoundTargets clause 1
Found 0 Counterexamples
Checked 20,973 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 1.232860e52 total values and 1.213597e52 total cases
Required 0:00:00.3 starting at 11:18:06 PM
11:18:06 PM : Checking ConditionalSoundTargets clause 2
11:18:06 PM : Computing facts for ConditionalSoundTargets clause 2 ...
11:18:06 PM : Translating ConditionalSoundTargets clause 2 ...
Completed translation of ConditionalSoundTargets clause 2
Required 0:00:01.7 starting at 11:18:06 PM
11:18:07 PM : Checking ConditionalSoundTargets clause 2 ...
Completed checking ConditionalSoundTargets clause 2
Found 0 Counterexamples
Checked 10,553 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 1.232860e52 total values and 1.213597e52 total cases
Required 0:00:00.2 starting at 11:18:07 PM
11:18:08 PM : Checking ConditionalSoundTargets clause 3

```
11:18:08 PM : Computing facts for ConditionalSoundTargets clause 3 ...
11:18:08 PM : Translating ConditionalSoundTargets clause 3 ...
Completed translation of ConditionalSoundTargets clause 3
Required 0:00:01.9 starting at 11:18:08 PM
11:18:10 PM : Checking ConditionalSoundTargets clause 3 ...
Completed checking ConditionalSoundTargets clause 3
Found 0 Counterexamples
Checked 524,226 values and 0 cases
Covered 100% of the total assignment space
Assignment space includes 1.232860e52 total values and 1.213597e52 total cases
Required 0:00:03.9 starting at 11:18:10 PM
11:18:13 PM : Completed checking ConditionalSoundTargets
Completed checking ConditionalSoundTargets
Found 0 Counterexamples
Checked 0 cases and 555,752 values
Assignment space includes 3.698580e52 total values and 3.640790e52 total cases
Required 0:00:04.4 starting at 11:18:04 PM
11:18:13 PM : Completed checking Claim ConditionalSoundTargets
Script hla-own-tr.script2 finshed
```