

Distributed Simulation of Hybrid Systems with AnyLogic and HLA

Andrei Borshchev, Yuri Karpov, Vladimir Kharitonov

Experimental Object Technologies www.xjtek.com, and
St. Petersburg Technical University, Russia

Introduction

A large class of systems being developed has both continuous time and discrete time behavior. In fact, any system that interacts with physical world falls in that class. Chemical, Automotive, Military, Aerospace are areas most frequently mentioned in this respect. To model such systems successfully and to get accurate and reliable results from simulation experiments one needs an executable language naturally describing hybrid behavior, and a simulation engine capable of simulating discrete events interleaved with continuous time processes. Additional problems arise with simulating hybrid systems in a distributed environment.

There is a number of tools, commercial and academic, capable of modeling and simulating systems with mixed discrete and continuous behavior (so called hybrid systems), for a good survey we refer to [2] and [9]. We believe that the most convenient way of hybrid system modeling is to specify continuous behavior as a set of algebraic-differential equations associated with a state of a state machine. When a state changes as a result of some discrete event, the continuous behavior may also change. In turn, a condition specified on continuously changing variables could trigger a state machine transition – so called change event. State machines run within objects that communicate in discrete way, e.g. by message passing, as well as by sharing continuous-time variables over unidirectional connections. Complex hybrid system modeling may require distributed simulation due to system complexity, performance and interoperability requirements, etc. Developed simulation should interoperate with other components, possibly created with different tools. This could be achieved by using some M&S standard. This would also allow creation of distributed simulations, where components run on different machines and different platforms [12]. We believe that High Level Architecture (HLA) for Modeling and Simulation developed by US DoD ([4], [5] and [6]) is the most suitable for this purpose.

In the paper we present AnyLogic, a tool for modeling and simulation of hybrid systems and a way of HLA support integration in the tool simulation engine [3]. To demonstrate AnyLogic ability to model and simulate hybrid systems, we present a simple example – two tanks system [9]. We examine problems aroused with distributed simulation of this system in AnyLogic using HLA.

The paper is organized as follows. Section 2 presents AnyLogic tool and its modeling language. A hybrid system example and its modeling in AnyLogic environment is described in section 3. Section 4 gives an overview of AnyLogic simulation engine integration into HLA. Distributed model of two-tanks system designed in HLA is also described here together with problems of hybrid system simulation in distributed environment. Section 5 concludes the discussion.

AnyLogic and Its Modeling Language

AnyLogic [1] architecture is shown in Fig. 1.

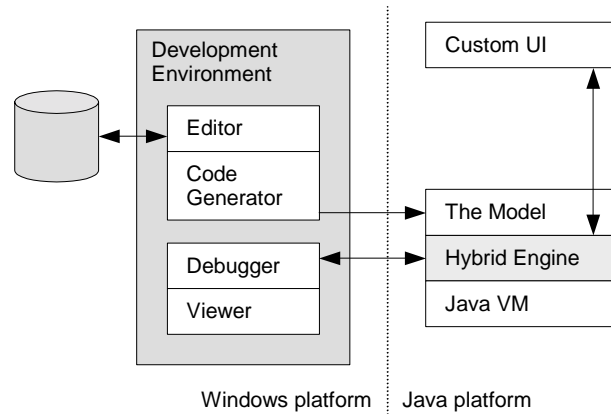


Figure 1 Architecture of AnyLogic Modeling and Simulation Environment

Windows-based Development Environment includes graphical model Editor and Code Generator that maps the model into Java code. The model runs on any Java platform on the top of AnyLogic Hybrid Engine. A running model exposes an interface to control its execution and to retrieve information via a text-based protocol over TCP/IP. That interface is used by Viewer and Debugger that runs on Java platform as well. The model supports connection of multiple clients from arbitrary (e.g. remote) locations.

We have chosen a subset of UML for Real Time as a modeling language, and extended it to incorporate continuous behavior. The language supports two types of UML diagrams: collaboration diagrams and statechart (state machine) diagrams with some changes. In collaboration diagrams we have added unidirectional continuous connections between objects (capsules in UML-RT) and the corresponding interface elements – input and output variables.

The main building block of a hybrid model is called active object. The object interface elements can be of two types: ports and variables. Objects interact by passing messages through ports, or by exposing continuous time variables one to another.

Object may encapsulate other objects, and so on to any depth. Encapsulated objects can export ports and variables to the container interface, see Fig. 2.

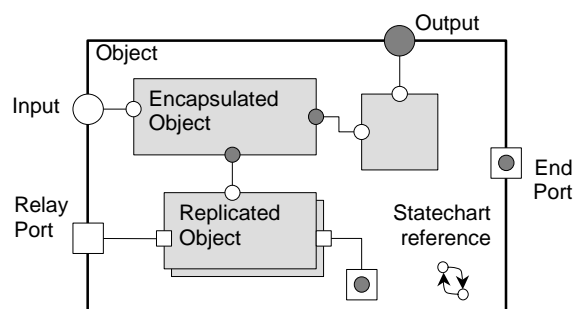


Figure 2 AnyLogic Structure Diagram extending UML-RT with continuous connections

An object may have multiple concurrent activities that share object local data and object interface. Activities can be created and destroyed at any moment of the model execution. An activity can be described by a Java function or by a (hybrid) statechart.

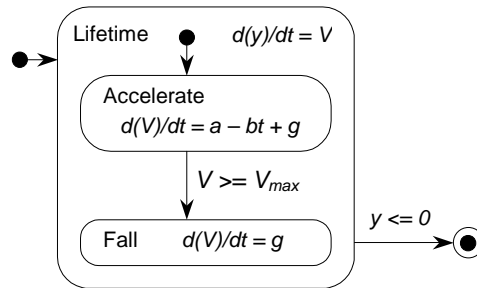


Figure 3 AnyLogic Hybrid Statechart

In addition to standard UML attributes of states and transitions, in hybrid statecharts one can associate a set of differential and algebraic equations with a simple and/or composite state of a statechart, and you can also specify a condition over continuously changing variables as a trigger of a transition. The currently active set of equations and triggers is defined by the current simple state and all its containers.

The example hybrid statechart in Fig. 3 is a simple model of an object that accelerates vertically up until it reaches the speed of V_{max} , and then falls under the impact of gravity until it touches the ground ($y \leq 0$), where it ceases to exist.

Hybrid system example

Consider a system consisting of two tanks and a controller (Fig 4). Three valves controls water injection in tank 1 ($v1$), water flow from tank 1 to tank 2 ($v2$), and water flow from tank 2 outside the system ($v3$). Controller tracks water level in both tanks (h_1 and h_2) and generates commands to open or close valves. The main task is to avoid droughts or overflows of tank 2. (AnyLogic demo with this and other examples is available from <http://www.xjtek.com>.)

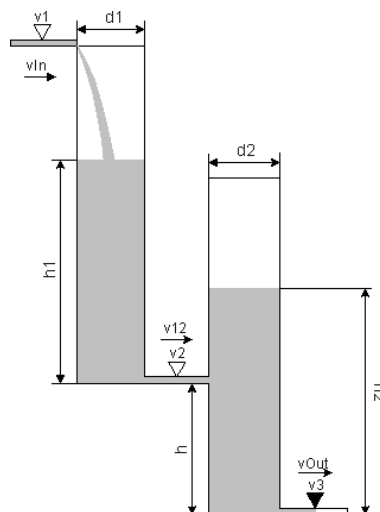


Figure 4 Two tanks system example

As we can see from the system description, there are two components: two tanks and the controller. Structure diagram of two tanks component of AnyLogic system model is presented in Fig. 5. Output variables $h1$ and $h2$ are used to expose water levels in tanks 1 and 2 respectively.

Ports $vXOn$ and $vXOff$ are used to receive commands for appropriate valves. Variables $k1$, $k2$, $p1$ and $p2$ are used in hybrid statecharts $trackV1$, and $trackV2$ to model water flow through valves $v2$ and $v3$ while they are in transit from opened to closed states and vice versa.

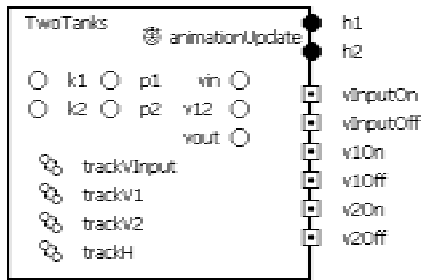


Figure 5 Two tanks component structure diagram

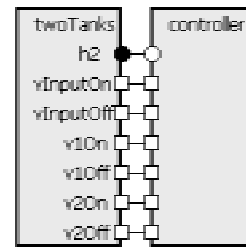


Figure 6 Overall system structure

Fig. 7 presents hybrid statechart *trackV1*. Controller component consists of one statechart implementing its logic (initially fill tanks and then track water level in tank 2), opening valve *v3* when *h2* goes below l and closing it when *h2* rises above l^+ . The overall system structure diagram is presented on Fig. 6.

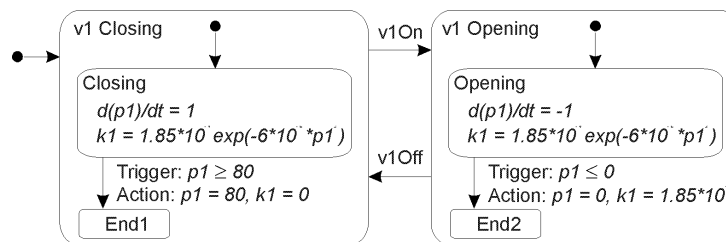


Figure 7 TrackV1 activity (hybrid statechart)

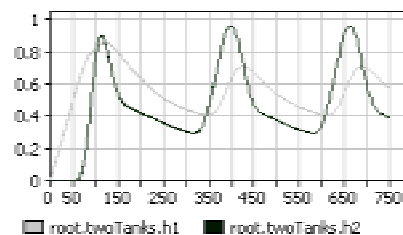


Figure 8 Simulation results of AnyLogic model of two tanks problem

Note the continuous variable connection on *h2* between components. Simulation results (water level in tank1 and tank2) of the system are presented in Fig. 8.

Distributed Simulation with HLA Support

The High Level Architecture (HLA) is a standard framework that supports simulations composed of different distributed simulation components. The HLA was developed by the Defense Modeling and Simulation Office (DMSO) of the Department of Defense (DoD) to meet the needs of defense-related projects, but it is now increasingly being used in other application areas [11]. The primary goal of such architecture is to facilitate simulation interoperability and reuse across a broad range of applications [7,8]. Recent adoption of the HLA as an IEEE standard will strengthen positions of this architecture among other modeling and simulation standards. There are examples of the successful creation of distributed simulations composed of components developed using different tools [10].

The HLA follows a framework approach and is defined by three major elements:

- Rules [4] govern the behavior of the overall distributed simulation (Federation) and their members (Federates);
- An Interface Specification [5] prescribes the interface between each federate and the Runtime Infrastructure (RTI), which provides communication and coordination services to the federates;

- An Object Model Template [6], which defines the way federations and federates have to be documented (using the Federation Object Model and the Simulation Object Model, respectively). Federations can be viewed as a contract between federates on how a common federation execution is intended to be run.

In simulation, HLA plays a role similar to one CORBA, COM+, etc. play in object oriented distributed software development.

Integrating Simulation Engine of AnyLogic and HLA

Integrating HLA support in AnyLogic will give the user the possibility of rapid creation of prototypes of component simulations (federates) and development of distributed simulations (federations) using convenient and powerful graphic environment of AnyLogic. When a prototype developed in AnyLogic proves its ability to deal with the problem it is intended to solve, one or more of the federates could be re-implemented using, for example, raw HLA interface on one of the high-performance languages such as C++. We believe that such approach for federation development could reduce the time and cost of simulation development and avoid many errors on early phases of the development process.

If we wish to wrap AnyLogic model in HLA federate which will be neither time regulating nor time constrained, then we should not add any special time coordination service calls to the simulation algorithm. In this case there will be no correspondence between local AnyLogic model time and federation time. All messages exchanged between AnyLogic federate and other federates are RO messages (delivered as soon as possible).

But if we wish to build time regulating and/or time constrained federate with AnyLogic, we should modify simulation engine to allow coordinated time advancements of all distributed simulation participants. The most general technique to achieve such coordination is using zero lookahead value (for time regulating federates) and Next Event Request Available (NERA) HLA Time Management service call. NERA(t) service allows delivery of all queued RO messages, grants time advancement to the time t (if no more TSO messages will be delivered with time stamp less than t) or to the time $t_1 < t$, where t_1 is the lowest time stamp of all scheduled TSO messages. Then it delivers this TSO message to the federate. Usage of NERA service call allows sending and/or receiving additional TSO messages scheduled at the current time and allows seamless integration of local AnyLogic and HLA simulation engines.

Simulating engines are often divided into two groups: open and closed engines.

Open engines provide special programmer interface for controlling model execution. User can attach some external module to this interface and have some control on model core.

Closed engines do not provide such interface. Model execution process is hidden from user code.

AnyLogic has been representing a close system for a long time [3]. No interface has been provided to user for controlling the process of model execution.

When integrating HLA module we have faced the problem of lack of such interface that would provide a user with ability for synchronizing model time with some global time clock – federation time (see chapter above).

We have developed universal interface that can be used by any user external module, for example by module binding some real world object to AnyLogic model. This interface is used in particular by HLA Support Module (HLA Support Module is described below) interconnecting AnyLogic and HLA federation. This interface is called StepHook.

AnyLogic StepHook Interface

AnyLogic provides user-accessible service for determination of time stamp of the next continuous or discrete event. Extension of AnyLogic simulation engine with such services allows integration of HLA support as an add-on package.

StepHook interface allows the user to put a hook on engine performing model time steps. Special method *Engine.setStepHook(com.xj.anylogic.StepHook)* is used to set a time step hook.

StepHook interface incorporates two methods:

1. double *nextEvent* (double time)

This method is called by the engine just before each time step. The single method parameter is a time when the next event is scheduled. It can be Double.POSITIVE_INFINITY if there are no events sched-

uled. The method implementation can return another time moment, that must not be later than the *time* moment, so the system clock will be adjusted only to this time point.

2. `void timeStepDone()`

The method is called by the engine right after the system clock has been adjusted to the time obtained from `nextEvent()`.

AnyLogic Federate. HLA Support Module

To facilitate federation interaction, special component – a part of AnyLogic model – has to route messages and synchronize local system clock to federation Global Time. Such component has been developed. At present time AnyLogic core interacts with Runtime Infrastructure through HLA Support Module (HSM). The described above *StepHook* interface is used as an interaction interface between HLA Support Module and AnyLogic Model Engine.

The structure of HLA federation with AnyLogic federate among other federates is shown in Figure 9.

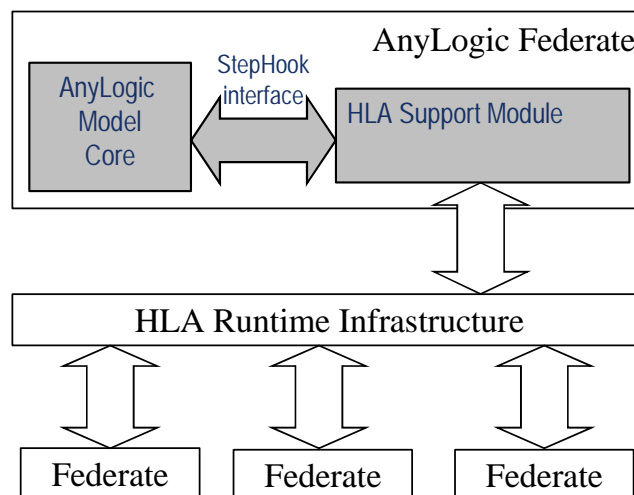


Figure 9 Integrating HLA and AnyLogic

HLA Support Module provides a range of services to AnyLogic model. User code can access HSM services to interact with other federates. The following services are provided at present time:

1. Federation Management (FM)
 - Federation execution create/join/resign/destroy
 - Federation-wide synchronization points
2. Declaration Management (DM)
 - Object classes publication/subscription
 - Interaction classes publication/subscription
 - Declaration advisories
3. Object Management (OM)
 - Object instances registration/discovery/deletion
 - Interactions sending/receiving
 - Attributes updates/reflection
 - Object advisories
4. Time Management (FM)
 - Time advancing
 - Regulating/constrained switches
 - Time related controls and queries

HLA integration in AnyLogic requires careful consideration. The most difficult problems arise in distributed simulation because of hybrid nature of simulated system components.

Distributed simulation of two tanks problem with AnyLogic

The developed AnyLogic HLA support module allowed us to build distributed simulation of described two tanks system model. System has been represented as one object class `TwoTanksSystem` with attributes `Tank1Level` and `Tank2Level`. Interaction class `ValveState` with parameters `Valve` and `IsOpen` allows federation participants to change valve states of the instance of the system. The HLA federation consists of three federates: two tanks simulator federate, controller federate, and viewer federate providing visualization of the process. Two tanks simulator federate publishes object class `TwoTanksSystem` with both attributes, creates and registers one instance of that class and updates its attributes. It also subscribes to `ValveState` interaction and translates it to the messages to appropriate ports (`vXOn` and `vXOff`). Controller federate subscribes to object class `TwoTanksSystem` with both attributes and later will discover object instance created by two tanks simulator. It also publishes interaction class `ValveState` to be able to send interactions of this class. AnyLogic models of system components have been wrapped by another `ActiveObjects` responsible for registration or discovery of instance of appropriate HLA object class, updating instance attribute values (periodically), and translating commands previously sent via ports to and from HLA interactions. Additional `ActiveObject` called `HLATimeAdvancer` has been added to every federate to synchronize local simulation engine time with federation time by requesting HLA RTI for time advancements as described in previous section.

Distributed simulation of the model shows overflows of tank 2. Because model logic has been left unchanged, the problem source is in breaking connections between two tanks and the controller (Fig 6). Connections between ports of the components have been represented as interactions between distributed components. Sending or receiving message to/from port is a discrete event, thus no information has been lost by such representation. But transmission of continuous time variable $h2$ only in discrete moments of time (periodic updates) with update period greater than some Δt will not allow controller properly react to the change in water level. Previous update may indicate normal level (below l^+), but the next one may show very high level or even overflow. This is an instance of more general sensitivity problem.

So, we are facing problem with exposition of continuous time variable in distributed simulation of hybrid system and detecting condition defined on it in another distributed component.

There are no significant conceptual problems with building distributed simulations of discrete event systems according to system state updates. Situation changes when one or more components have continuous time or mixed (hybrid) behavior, which they want to expose to other components.

The problem is to represent hybrid system as a discrete event system at the level of distributed components interactions.

Three general approaches could be proposed: value polling, sampling along time or value axis, and an approach which provides ability for one component to define a predicate on a variable, which is to be evaluated locally at another component along with notification when such event occurs.

First two mentioned approaches are quite obvious. Their disadvantage is that they can't solve the problem of guaranteed correct detection of conditions defined over continuously changing interface variables. Below we propose an approach, called Remote Predicate Evaluation, which can cope with those difficulties.

Detecting conditions defined over continuously changing interface variables. Remote Predicate Evaluation (RPE) technique

Polling and sampling update methods are good enough when we need to monitor behavior of components, e.g. for building external viewers, statistic collecting, tracking objects, and other situation awareness needs. However, these update methods are not very good for detecting conditions defined over continuously changing interface variables (like $h2$ in the distributed two tanks system).

In hybrid system there are two basic possible interconnections between interacting objects (see Figure 10).

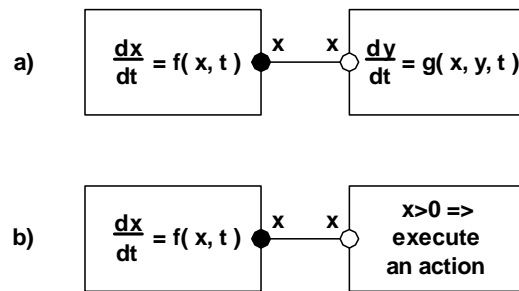


Figure 10 Hybrid System Components Interconnection

First, there are situations when the variable, being exposed by one object is used in differential equations of another object (see Figure 10, a). Second, there are situations when we are interested in the value of a predicate (condition) defined on a continuous time variable (see Figure 10, b). This, for example, may affect discrete state of the system or trigger some actions associated with this event in other components.

Remote Predicate Evaluation (RPE) is the method when such predicate could be checked locally within the component, which exposes the variable(s) while solving algebraic-differential equations. It could provide required accuracy in determining the moment of time when this event occurs. Besides it allows distributed model designer to lower the probability of sensitivity problem appearance and minimizes overhead caused by frequent variable value updates (only the fact of event detection is announced to other interested components). Finite amount of information is required to transfer both a predicate and a notification over the network.

It could be recommended to design distributed simulation in the way that components “encapsulate” their continuous behavior, exposing continuous time variables (attributes) only for the needs of situation awareness, visualization and remote statistic collection at relatively low rate. Detection of all required events identified during simulation (federation) design and development is then performed inside the component and the corresponding notification is sent to all interested components.

Sometimes, however, simulation components model devices with analog output, which is continuous by its nature (e.g., electrical current or voltage interface). A model designer may not know a priori which conditions will be interesting for components connected to this device during distributed simulation execution. In this case a mechanism for dynamic creation and modification of predicates on output variables can be implemented.

If the designer has a priori knowledge of the form of the predicate, she could parameterize it and allow other components to change parameters during simulation execution, tuning this component for their needs. In HLA this effect could be achieved, for example, by declaring attributes with transferable ownership representing predicate parameters. Or it could be implemented using specialized interaction exchange protocol. As we describe below, evolving the idea, we suggest designing predicates as remotely loaded agents, so almost no limitations to predicate form are applied.

RPE Applicability to Two Tanks Model

The latter method was implemented with prototype HLA support add-on to AnyLogic for distributed simulation of above-mentioned two tanks problem. Controller tracks value of h_2 and when it reaches some dangerously high level L^+ , it commands appropriate valve to open. In other words, controller component defines predicate on continuous variable h_2 in the form $h_2 > L^+$. Since h_2 is modeled in another component, the h_2 update method can directly affect the time delay before open valve command is issued and thus can lead to tank 2 overflow (this is just what we’ve got during distributed simulation using sampling). The same system simulated as a single AnyLogic model does not show such overflow. Here we can see that distributed simulation of hybrid system can show wrong results just because of continuous variable update delay.

The solution of this problem is to allow controller federate define L^+ value for the two tanks federate. We have done this by defining HLA interaction with parameter specifying L^+ value. After interaction reception two tanks federate changes parameter of the predicate on h_2 ($h_2 > L^+$) and evaluates it during solving system of differential equations. After predicate becomes true (an event detected), the federate updates values for h_1 and h_2 and the controller detect this event with minimal possible error. As the predicate is evaluated “remotely” by the federate simulating continuous time variable, we call this method Re-

remote Predicate Evaluation. It showed its ability to deal with the described problem for this particular example of distributed simulation.

RPE has several obvious limitations. For example, it cannot help if we have predicate on more than one continuous time variable simulated by different distributed components. For this case revision of model partitioning into distributed components could be advised. Obviously, subcomponents tightly coupled by continuous variables should be placed in the same distributed component.

RPE Implementation

Remote Predicate Evaluation technology at present time is being integrated to AnyLogic tool. We have faced the problem of transferring the predicates to remote model. Different approaches has been reviewed. After detailed analysis we came to conclusion to design remote predicates as remotely loaded code – remotely loaded java classes.

We decided to use the popular idea of agents, developed to comply the special interface that would be provided by AnyLogic.

Using HLA to transfer RPE agent code

To transfer agent code different technologies (different existing protocols) can be used. We suggest using RTI services to transfer RPE agent code from one federate to another. In particular we came to conclusion to use HLA interactions to transfer serialized agent java code.

This solution is bind only to RTI services, not to some special protocol. Thus, we do not limit the network environment in any way.

This special interaction (RPE interaction) has to be described in Federation Object Model Template. All federates implementing RPE techniques must publish and subscribe to interactions of this type.

On receive RPE interaction (with agent code inside), HLA Support Module doesn't transfer it to AnyLogic model engine, it passes such interaction to special Remote Predicate Evaluator module.

Module Interaction

To support remote predicate agent loading special module is being developed. We have called this module Remote Predicate Evaluator module. This module interacts both with AnyLogic model engine and HLA Support Module.

Module interconnection is shown in Figure 11.

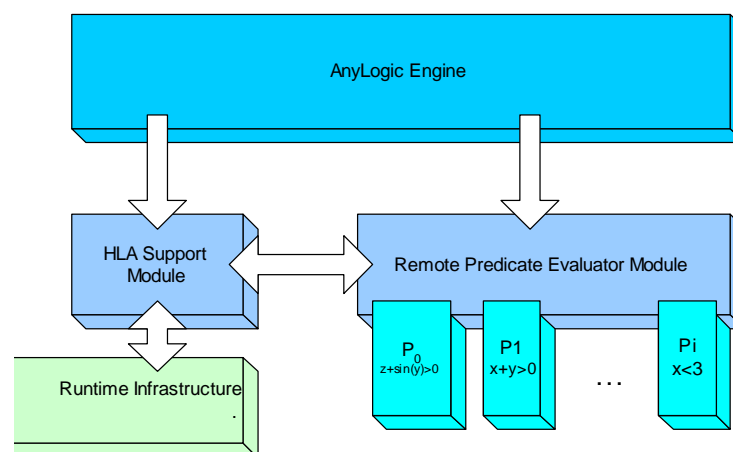


Figure 11 RPE Module Interactions with Other System Modules

The process of predicate loading is described below:

1. Predicate code, after having been transferred through network is passed by RTI to HLA Support Module as interaction of some special type. This special interaction serves as a container for predicate code.
2. HLA Support Module detects the type of interaction. If type equals RPE interaction, HLA Support Module doesn't pass it to AnyLogic Model, but forwards it to RPE module.
3. RPE module loads the transferred class to a predefined vector of remote predicates.

Each model step all predicates in predicate vector are being evaluated. RPE module analyses each predicate state. If any predicate triggers its state, RPE module generates local event, which is handled by AnyLogic engine.

Such event is associated with engine procedure that reflects all model attributes to federation.

Remote Predicate Evaluator Interface

Remotely loaded predicate module must be a successor of some abstract class. This class has two predetermined methods.

1. *boolean evaluateStep(double[] vector, double t)*

double[] vector – phase vector, a vector of federate continuous variables (this parameter is generated by AnyLogic engine)

double t – current AnyLogic federate local time.

EvaluateStep() method is executed each integration step. This method should return true if predicate turns true (false in opposite case).

2. *boolean evaluateEvent(double[] vector, double t, object[] objects)*

double[] vector – a vector of federate continuous variables.

double t – current federate local time.

object[] objects – a vector of variables, published by federate to federation, except continuous variables.

This method is executed each local event time. This method can contain special additional predicate code, which is executed only at local event time. Additional information (*objects[]*) is passed to this method in comparison to *evaluateStep()*. Vector *objects[]* contains variables that may change only at local event time. These variables (for example of type *String*) are not continuous variables, but these variables can be included in complex predicate expression.

This method should return true if predicate turns true (false in opposite case).

Benefits

The suggested solution being developed is ultimately fast. No string parsing is needed when checking predicate triggering. Solution uses precompiled java code.

No recompilation is needed on agent loading. Agents are precompiled on remote federates.

Agents can be loaded and unloaded dynamically.

No limitations on remotely loaded predicate are applied. User can write almost any code. Agent can represent predicate of any complexity. For example, agent can save and analyze the history of behavior of federate it has been loaded to.

Agents can be parameterized. Some remote federate can send agent to local federate, wait it executes some time, unload it, and resend agent with another parameters.

Drawback and Limitations

The first limitation that must be applied to agent code is that remotely loaded agent has to be successor of some abstract class. Remotely loaded agent must represent predetermined interface to interact with Remote Predicate Evaluator module.

The second limitation is restriction to agent speed. Agent code is evaluated almost each time model engine make an integration step. That means that agent code is restricted to be very fast. It is not a problem if we speak about ordinary predicates representing just algebraic expressions. But it may be a problem

when developing complex predicates that may, for example predicates analyzing the history of federate variables behavior.

Conclusion

Hybrid statemachines approach implemented in AnyLogic modeling and simulation environment is a powerful and convenient formalism to describe behavior of the real world systems. AnyLogic itself is a very flexible tool, it is essentially an environment for programming on Java with modeled system visual specification support in terms of simulation class library. This property of AnyLogic makes it relatively easy to develop HLA support extensions and enable components created with this tool participate in distributed simulations.

There is a single common standard of distributed simulation in military domain. For a long time there was no such standard in civil domain. Recent HLA adoption as IEEE standard has improved situation. It can help with interoperability and reuse of simulations created with different tools. HLA is suitable for discrete event systems, but problems arise with distributed hybrid systems modeling with HLA. Distributed execution of components connected on continuous time variables and detection of events related to them may lead models to demonstrate results which differs drastically from the simulation of the same system locally. One example of such problem has been demonstrated and solution named Remote Predicates Evaluation has been proposed.

RPE techniques integration with HLA standard has been proposed. Of course, RPE techniques being developed represents some superstructure upon HLA standard. And what is more, current technology is limited to Java federates, because RPE agent code, being transferred by Runtime Infrastructure as an interaction represents Java serialized object. But there are many abilities of RPE technique improvement, we continue working on.

References

1. AnyLogic 4.0 User Manual. Available from <http://www.xjtek.com/products/anylogic/40/>
2. Astrom, J., Emqvist, H., Mattson, S.E.: Evolution of Continuous-time Modeling and Simulation. In: Proceedings of the 12th European Simulation Multiconference, ESM'98, Manchester, UK (1998)
3. Borshchev, A.V., Kolesov, Yu.B., Senichenkov, Yu.B.: Java Engine for UML Based Hybrid State Machines. In: Proceedings of the Winter Simulation Conference 2000, WSC'00, December 10-13, Orlando, FL, USA (2000)
4. IEEE P1516, Standard [for] Modeling and Simulation (M&S) High-Level Architecture (HLA) – High Level Architecture Rules.
5. IEEE P1516.1, Standard [for] Modeling and Simulation (M&S) High-Level Architecture (HLA) - Federate Interface Specification
6. IEEE P1516.2, Standard [for] Modeling and Simulation (M&S) High-Level Architecture (HLA) - Object Model Template (OMT)
7. Li, B., Li, X., Chai, X., Qing, D.: A HLA Integrated Development Environment. Society for Computer Simulation Conference, (July 1999)
8. Allen, R., Garlan, D., Ivers, J.: Formal Modeling and Analysis of the HLA Component Integration Standard. In: Proceedings of the 6th International Symposium on the Foundations of Software Engineering (FSE-6) (1998)
9. Kovalewski et al.: A Case Study in Tool-aided Analysis of Discretely Controlled Continuous Systems: the Two Tanks Problem. In 5th International Workshop on Hybrid Systems, Notre Dam, USA (1997)
10. Steffen Straßburger: On The HLA-based Coupling of Simulation Tools. Available online from <http://isgsim1.cs.uni-magdeburg.de/hla/paper/HLASimTools.pdf>
11. Thomas Schulze, Steffen Straßburger, Ulrich Klein: Migration of HLA into Civil Domains: Solutions and Prototypes for Transportation Applications. Available online from <http://isgsim1.cs.uni-magdeburg.de/hla/paper/S7305-05.pdf>
12. Günther Seliger, Dirk Krützfeldt, Peter Lorenz, and Steffen Straßburger: On the HLA- and Internet-based Coupling of Commercial Simulation Tools for Production Networks. Available online from <http://isgsim1.cs.uni-magdeburg.de/hla/paper/ICWMS99.pdf>