EODiSP – AN OPEN AND DISTRIBUTED SIMULATION PLATFORM

9th International Workshop on Simulation for European Space Programmes
SESP 2006

6-8 November 2006 at ESTEC,
Noordwijk, the Netherlands

I. Birrer [1], B. Carnicero-Dominguez [2], M. Egli[3], B. Carnicero-Dominguez [2], A. Pasetti[1,3]

[1]*P&P Software GmbH*
C/O ETH Zentrum, Physikstrasse 3, Zurich CH-8092, Switzerland
{Birrer, Pasetti}@pnp-software.com

[2]*ESA-Estec*
PO Box 299, Noordwijk, The Netherlands
{ Bernardo.Carnicero.Dominguez, Joachim.Fuchs}@esa.int

[3]*Institute fur Automatik, ETH-Zurich*
Physikstrasse 3, Zurich CH-8092, Switzerland
eglimi@control.ee.ethz.ch

INTRODUCTION

This paper describes the EODiSP (*EODiSP Open and Distributed Simulation Platform*). The EODiSP is a generic platform to support the development and operation of distributed simulators that integrate HLA-compliant simulation packages. It was developed for the European Space Agency under contract number 18833/05/NL/AR. Its development was led by P&P Software GmbH (CH) with the Institute fur Automatik of the ETH Zurich as subcontractor.

The EODiSP is specifically aimed at supporting the development of end-to-end simulators for earth-observation satellite missions but is more generally suitable for simulators that are built by integrating heterogeneous and distributed simulation packages that interact by sending data to, and receiving data from each other. The EODiSP is built as a partial implementation of the High-Level Architecture (HLA). The HLA is the most widely used simulation architecture. It is defined by an IEEE standard as a bundle of services that support various aspects of a simulation. The EODiSP implements the subset of HLA services required to support data-driven simulations. The full complement of HLA services may be implemented at a later date.

OBJECTIVES

The EODiSP is a generic platform to support the development and operation of distributed simulations. An EODiSP simulation is built by integrating a set of *simulation packages* with the EODiSP infrastructure. A simulation package is a piece of software that implements part of the functionalities required for an end-to-end simulation and that is delivered as a single unit. A simulation package encapsulates one or more simulation models.

The *simulation models* normally take the forms of algorithms, possibly defined in some modelling environment such as Matlab. The simulation packages implement the simulation model algorithms in software. Simulation packages can take a variety of forms: source code in a high-level language, binary level executable, macros in an excel spreadsheet, etc.

With the EODiSP approach, users are expected to define the simulation models required for a simulation and to encapsulate them in simulation packages. The EODiSP provides the infrastructure to allow the simulation packages to interact together over a possibly distributed network. The EODiSP is specifically intended to allow the integration of *heterogeneous* and *distributed* simulation packages.

AN HLA-BASED PLATFORM

The EODiSP is built as a partial implementation of the *High-Level Architecture* or HLA [1]. The HLA is the most widely used simulation architecture. The HLA is defined as an IEEE standard [2]. Adherence to the standard allows interoperability and reusability of simulation packages.

The HLA is defined as a bundle of services that support various aspects of a simulation. In general, a particular simulator will only need a subset of all the services defined by the HLA. At present, the EODiSP only supports a subset of the HLA services. These services are aimed at supporting data-driven simulations. It is expected that more HLA services will be supported in the future.

HLA TERMINOLOGY

The HLA standard defines a special terminology for describing HLA-based simulations. This terminology is important for understanding the material presented in this paper.

An *HLA Federate* is the basic unit of composition of an HLA-based simulation. A federate encapsulate an application that participates in an HLA-based simulation. In terms of the EODiSP terminology, an HLA federate encapsulates a simulation package. Note that a simulation package is a pre-defined piece of software that has been created prior to the definition of the simulation. It normally cannot be directly integrated in the EODiSP because its external interfaces do not comply with the HLA standard. Such a simulation package must normally be *wrapped* to be made HLA-compliant. Wrapping transforms the simulation package into an HLA federate ready for integration with the EODiSP platform.

An HLA federate is formally described by a *Simulation Object Model* or SOM. The SOM is an XML-based file that describes the interface of a federate in terms the data it generates as outputs and of the data it requires as inputs.

An *HLA Federation* is a set of federates that are used to build a simulation. An HLA federation is formally described by a *Federation Object Model* or FOM. The FOM describes how the federates in a federation are connected to each other. The actual operation of an HLA federation is called the HLA Federation Execution.

The HLA defines two mechanisms to allow federates to coordinate their activities. *Synchronization points* allow sets of federates in the same federation to synchronize their activities. *Interactions* allow federates to perform actions that have an effect on other federates in the same federation. The EODiSP predefined five synchronization points and one interaction. Developers may defined additional synchronization points and interactions that are specific to their simulation.

The HLA defines the *Runtime Infrastructure* or RTI. The RTI is a component that provides the generic infrastructure through which federates in a federation execution can exchange data.

SIMULATION INTERFACES

The federates that take part in an EODiSP simulation are characterized by their *interfaces*. The structure of these interfaces is in turn defined by the HLA standard. At their most basic, the interface defines:

- The data the federate needs as input and
- The data the federate generates as its output

The HLA standard implies a publish-subscribe architecture where federates publish the data they generate as outputs and subscribe to the data they need as input. A federate can only subscribe to data that are published by other federates in the same federation. The HLA standard supports both simple primitive types (floats, integers, booleans, etc) and more complex structure-like types. Some basic data types are defined by the standard but users can define new user-specific data types.

The main advantage of an interface-based environment is that it is easy to replace a simulation package with another simulation package. If the new simulation package has the same interface as the old one, then the change of simulation package has no impact on the rest of the simulation.

Figure 1 illustrates the publish-subscribe architecture mandated by the HLA and adopted by the EODiSP. The figure shows a simulation where three simulation packages exchange data. Each simulation package publishes its outputs and links its inputs to the outputs published by other packages. The data that are published and subscribed to by a certain simulation package define that package's interface. The term 'data' here can designate either a single value of primitive type or an instance of a complex data structure.

The figure shows the simulation package inputs and outputs being directly connected to each other. This is a logical architecture. The physical architecture may be very different because the transfer of data from a package output to a package input may not be direct (it may, for instance, be routed over a distribution network). This data shuffling however is performed by the EODiSP infrastructure invisibly to the simulation packages.
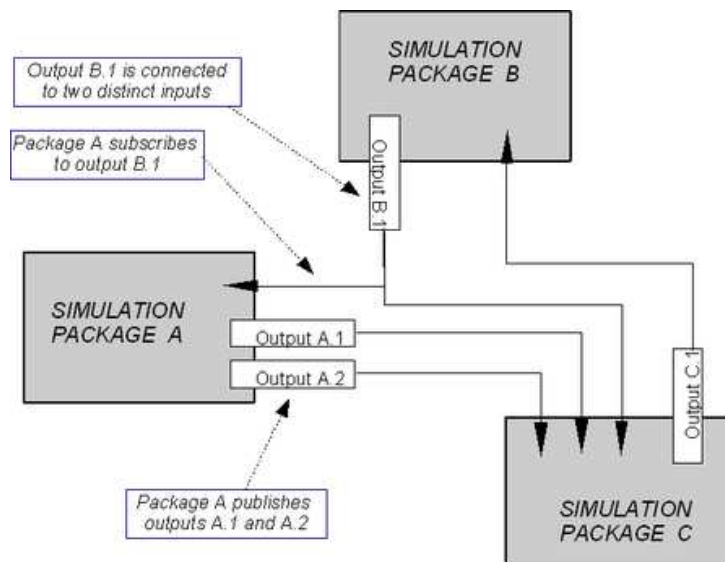
Fig. 1: Publish-Subscribe Architecture

WRAPPERS

The EODiSP is intended to allow the integration of heterogeneous simulation packages. The EODiSP provides an infrastructure to connect them together and to let them interact with each other to implement a complete simulation.

The EODiSP is built as an implementation of the HLA standard. The HLA defines the interface between the entities participating in a simulation (the HLA federates) and the simulation infrastructure. Hence, in order to be integrated with the EODiSP infrastructure, simulation packages must comply with this HLA-defined interface.

It is unrealistic to expect all developers of simulation packages to provide HLA-compliant packages. For this reason, the EODiSP supports the concept of *wrapper*. A wrapper is a piece of code the transforms a given simulation package into an HLA-compliant federate. The wrapper typically adapts the external interface of a simulation package to make it comply with the HLA requirements.

Wrappers obviously depend on the structure of the model they wrap. For this reason, it is not possible to provide a 'universal wrapper'. The skeleton of the wrapper - the part of the wrapper that interfaces to the EODiSP infrastructure - does however have a fixed structure. The EODiSP accordingly provides an application that can automatically generate this skeleton. This application is called the *HLA Wrapper Generator*. The generator takes as an input an XML-based description of the HLA interface of the target package and automatically generates the Java source code that implements the wrapper's skeleton.

Although it is not possible to define a fixed structure for a generic HLA wrapper, it will often happen that wrappers that are targeted at similar models to be used in a certain simulator (or, perhaps, in a family of related simulators) do have a fixed structure. In such cases, it may be advantageous to develop a domain-specific wrapper generator.

The wrapper generators are an example of an *EODiSP support application*. Prior to running an EODiSP simulation, a number of off-line tasks need to be performed. Some of these tasks can be partially or fully automated. An EODiSP support application is an application that helps users in performing these off-line tasks. At present the HLA Wrapper Generator is the only support application provided by the EODiSP. Other support applications may be provided in the future.

SIMULATION PARADIGM

The HLA supports a wide range of simulation paradigms (data driven, time triggered, event driven, etc). The current implementation of the EODiSP privileges the *data-driven paradigm*. In a data-driven simulation each simulation package is characterised by the data it produces and by the data it generates. Control flow is determined by the arrival of data: a simulation package is triggered as soon as all its input data have become available. The architecture of a simulation is defined by linking data sources (the producers of the data) to data sinks (the consumers of the data).

In accordance with the HLA approach, the EODiSP requires that each simulation package participating in a simulation declare which data it publishes (i.e. which data it generates and makes available to other simulation packages) and

which data it subscribe to (i.e. which data it requires from other simulation packages). The simulation can only be executed when all simulation packages know where to find the data they consume.

The type of simulation packages and the type of data they exchange is of course specific to each simulation. The EODiSP provides a reusable framework that implements the mechanisms for shuffling the data between simulation packages and for triggering the simulation packages. Data shuffling can be done over a distribution network (in case of a distributed simulation) and across language and operating system barriers (in case of simulation packages that are implemented in different languages and run on different platforms).

SIMULATION RUNS AND SIMULATION EXPERIMENTS

In simulation systems, a distinction is often made between a simulation run and a simulation experiment. A *simulation experiment* is a set of *simulation runs* executed in sequence with different configurations. A configuration is defined by a set of initialization files. An initialization file is a file that is read by a federate during the federate initialization phase. In the EODiSP, a federate can have at most one initialization file.

The EODiSP allows users to perform simulation experiments. Note that if a user wishes to execute a single simulation run, he can configure a simulation experiment with just one simulation run included.

DISTRIBUTES SIMULATION PACKAGES

The EODiSP supports the integration of physically distributed simulation packages. The EODiSP allows simulation packages residing on a network of distributed computers to be linked together to participate in a simulation. This allows developers of simulation packages to implement their models on their platform of choice (Unix, Linux, Windows, etc) while preserving the interoperability with other models.

The EODiSP distribution infrastructure is built on the Internet. Hence, no special communication hardware or software is required on the nodes that participate in a distributed EODiSP simulation.

The EODiSP distribution infrastructure is designed to by-pass firewall protections. This allows simulation packages that reside behind company firewalls to participate in an EODiSP simulation. The overhead that is added by the EODiSP distribution infrastructure is minimal in most cases.

The EODiSP can also be used to instantiate non-distributed simulations. Obviously, in this case, there is no overhead connected to the potentially distributed nature of the EODiSP.

CONCEPTUAL STRUCTURE OF AN EODISP SIMULATOR

The EODiSP is a generic platform that users must customize to build a specific simulator. Figure 2 below shows the conceptual structure of a simulator built using the EODiSP. The dark component at the centre of the figure represent the EODiSP framework. This is the generic simulation infrastructure that provides the mechanisms through which the simulation packages exchange data and control information. This infrastructure is entirely generic. In order to construct a simulation application, it has to be customized with the simulation packages that encapsulate the models that are required to run the simulation.

The simulation packages are represented in the figure by the light boxes. The process of customization of the EODiSP framework is schematically shown in the figure as the plugging of the simulation packages into the EODiSP infrastructure.

In some cases, the simulation packages are local in the sense that they reside on the same physical platform as the EODiSP framework. In the figure, this is the case of the simulation packages marked A and B. In other cases, the simulation packages are remote in the sense that they are designed to run on a platform different from the platform on which the EODiSP framework is located. In the figure, this is the case of the simulation packages marked C and D. Remote packages are connected to the EODiSP core through a distribution infrastructure.

The process of customisation of the EODiSP framework takes the form of a plugging of the simulation packages into the EODiSP core. The plugging is only possible if the simulation package has an external interface that is compatible with the EODiSP interface. When this is not the case, then the simulation package must be embedded within a wrapper component that performs a translation from the package interface to the interface required by the EODiSP framework. In the figure, this is the case of the simulation package marked as D. Note that wrapping can also be used to resolve other types of incompatibilities such as the case of the EODiSP framework and the simulation packages being implemented in different languages or compiled with different compilers.
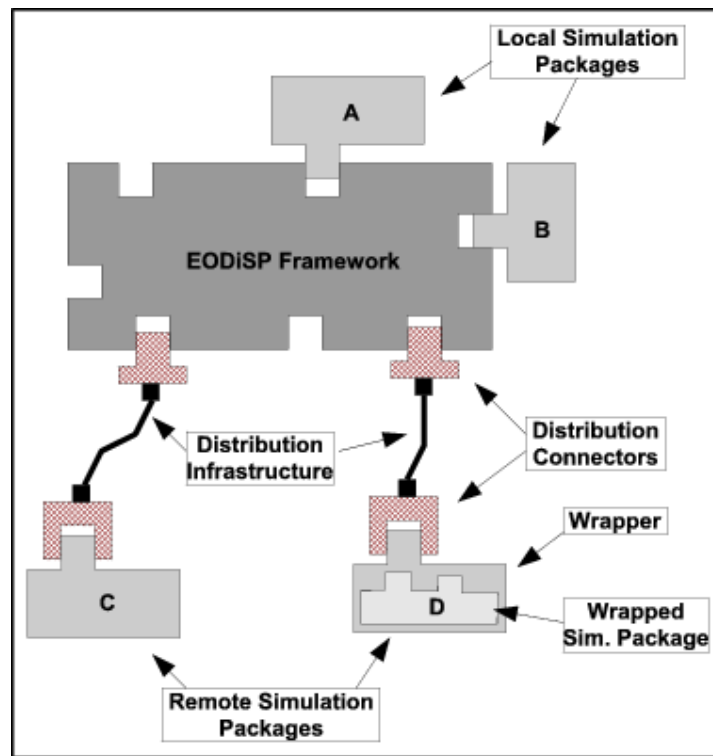
Fig. 2: Conceptual Simulator Structure

HLA STRUCTURE OF AN EODISP SIMULATOR

The previous section described the conceptual structure of a simulator built by instantiating the EODiSP. This section describes how this conceptual structure is implemented on top of the HLA.

Figure 2 can be re-cast as shown in figure 3 below. The simulation packages taking part in a simulation are now shown as HLA federates. The HLA defines the interfaces to which the simulation packages must conform in order to participate in an EODiSP simulation. When this is not the case (simulation package D in the figure), a wrapper is used.

The EODiSP framework is implemented as the HLA Runtime Infrastructure (RTI). The RTI is a component defined by the HLA that provides the generic infrastructure for an HLA simulation. It in particular keeps track of all attributes that are published by all federates in a simulation and is responsible for notifying registered federates whenever an attribute's value is updated.

As indicated in the figure, two categories of federates can be recognized. The first category are the user-defined federates. These are the federates that encapsulate the simulation packages provided by the user. These federates are normally specific to a particular simulation and must be entirely defined by the user. The second category are federates that are pre-defined by the EODiSP. These federates are the same in all EODiSP simulations and are provided with the EODiSP environment.

There are two pre-defined federates in an EODiSP simulation. The first one implements the management object model or MOM. The MOM is defined by the HLA standard. It is intended to make available to other federates information about a running federation. For this purpose, it publishes attributes that define items like: the number of federates that have currently joined the federation, the current version of the RTI, the host on which a certain federate is running, the number of updates made by a certain federate, etc. The MOM retrieves all this information from the RTI. The MOM can thus be seen as the bridge between the RTI and the user-defined federates: it is the means through which the user-defined federates can acquire internal RTI information.

The functionalities to be implemented by the RTI and the MOM are fully defined by the HLA. These functionalities represent a minimal core that must be provided by all HLA-compliant simulation environments. The EODiSP, like most HLA-based simulation environments, however, provides some additional functionalities. In keeping with standard practice in the HLA community, the EODiSP implements them in a so-called control federate. The control federate is the second pre-defined federate provided by the EODiSP. Unlike the MOM, it is not defined by the HLA and its implementation and interface are specific to the EODiSP. All EODiSP simulations must, however, include it. In the EODiSP, the control federate is responsible for managing synchronization points and interactions (see discussion of the

simulation lifecycle below) and for acting as a bridge between the simulation manager application (see the discussion of the EODiSP user interface) and the simulation infrastructure.
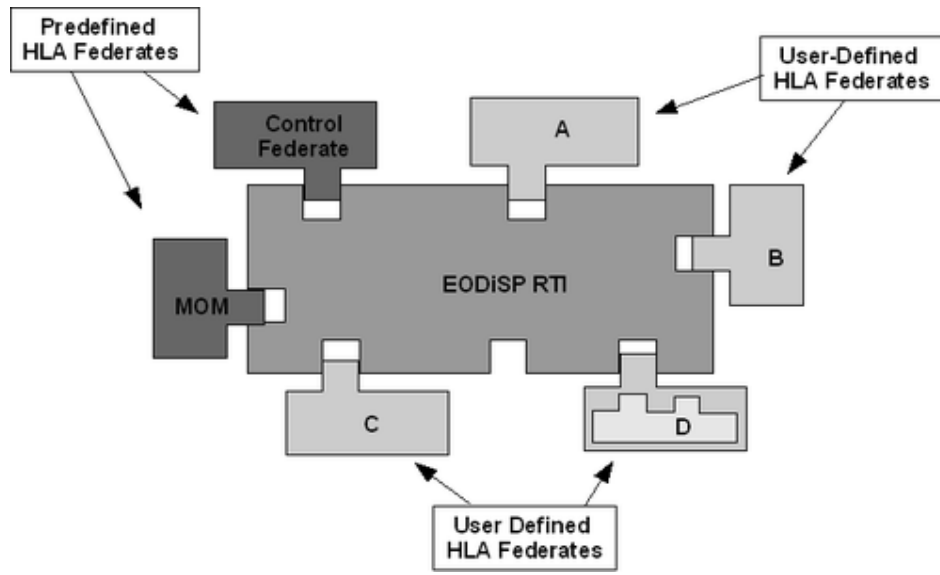
Fig. 3: HLA Simulator Structure

The distribution part of the EODiSP is implemented within the RTI. The federates see a local RTI interface and the RTI is responsible for routing attribute update notifications over a distribution infrastructure. For this purpose, and as shown in the figure 4, the RTI is split into two kinds of components: the Local RTI Component or LRC and the Central RTI Component or CRC. The LRC is a local representative of the RTI that directly interfaces with a federate. The LRC's are connected over a distribution infrastructure to the CRC that acts as a centralized router.
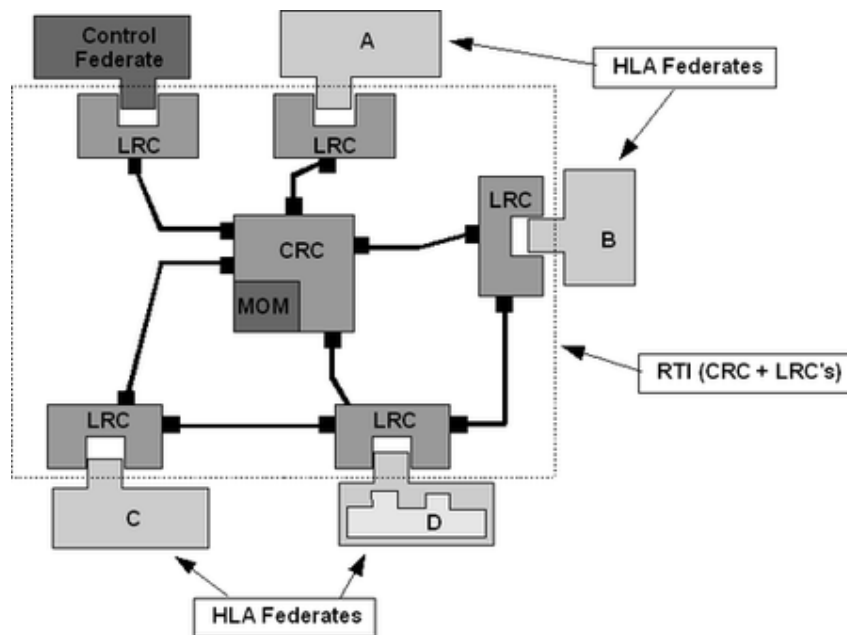
Fig. 4: HLA Simulator Structure

As shown in the figure, the MOM federate has a special position. Instead of being connected to an LRC, like all other federates, it is directly embedded within the CRC. This is because the function of the MOM is to make available to other federates information about the internal state of the CRC. It therefore makes sense to integrate its federate tightly with the CRC. The other federates, however, see it as an ordinary HLA federate.

It is useful to distinguish two types of communication channels within the RTI. Control channels are established between an LRC's and the CRC. These links are used to transfer typically short messages that describe changes in the

internal state of the LRC's or of the CRC. Data channels are instead used to transfer potentially large amounts of data from one LRC to another. In order to avoid cluttering, the figure only shows two such data channels (from federate C to D and from federate D to B) but in reality whenever federate X registers interest in an attribute owned by federate Y, then a data channel is set up from Y to X since data will have to be transferred from Y to X. Thus data transfers are designed to take the shortest route from the source of the data directly to the user of the data, without having to go through the CRC.

From an implementation point of view, it is important to point out that the EODiSP runs each LRC-Federate pair in a dedicated Java Virtual Machine (JVM). The advantage of this choice is that it isolates each federate in its own executable. This provides a natural and effective fault tolerance for a simulation as a whole: if one federate crashes, then no other federate needs be affected and the federation within which the federate is embedded may in principle continue execution (albeit in a degraded mode). The drawback of this choice is the overhead associated with starting and running several JVMs on the same node. This overhead however is regarded as acceptable in most practical situations where the number of federates running on the same node is expected not to exceed 20-30.

There is another related implementation issue that may directly affect users of the EODiSP. This concerns the threading policy that is adopted by the EODiSP for updating federate attributes. Essentially, the behaviour of a federate in an EODiSP simulation is driven by the arrival of notifications of updates of attribute values. Since the update notifications arrive asynchronously and the time required to process them varies from federate to federate (and, indeed, from attribute to attribute), the question arises of how the update requests should be handled within the EODiSP.

The policy adopted in the EODiSP is best described with the help of figure 5. The figure shows a federate which subscribes to three attributes. As discussed above, the federate receives the update notifications through the LRC and the LRC runs as a self-standing application on a dedicated JVM. Within this JVM, the EODiSP allocates a dedicated thread to each new update request. All update request threads have the same priority and they are configured to run to completion (the federate is encapsulated in a Java synchronized object). In other words, if an update request is received while a previous update request is still being processed, the later request is suspended. This ensures the integrity of the internal federate data. The model code will normally run in response to the update of certain attributes. This means that the model code will run on one of the update request threads. To exemplify, and with reference to the figure, consider the case where the model code only runs when all three attributes have been updated. In such a case, the model code would run on the thread that is associated to the attribute that is updated last.
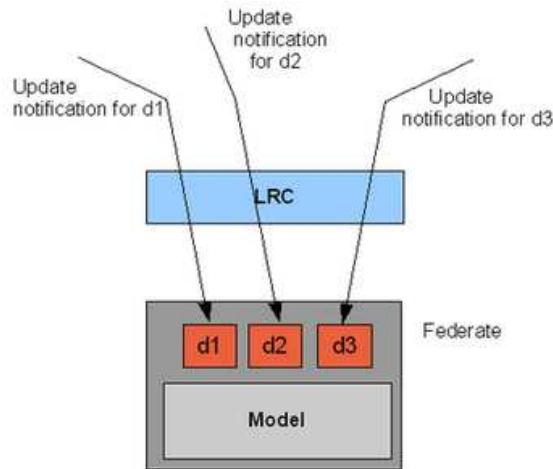

Fig. 5: HLA Attribute Update

EODISP USER INTERFACE

The EODiSP is operated through two GUI-based applications: the *simulation manager* and the *model manager*. The simulation manager can only be instantiated once for each simulation. It controls the simulation environment (the HLA RTI) and acts as the simulation server. The model manager application is instantiated at least once for every distribution node that participates in a simulation. A model manager application controls a set of simulation packages encapsulated in HLA federates that reside on the same node. The model managers act as clients to the simulation manager server.

Two categories of users participate in an EODiSP simulation. The first type of user is the *simulation owner*. This is the person who is in overall control of a complete simulation. The simulation owner decides how the simulation packages encapsulating the simulation models should be configured and when a simulation should start and terminate. The simulation owner interacts with the EODiSP through the simulation manager application.

The second type of user is the *model owner*. The model owner is a person in charge of one or more simulation models. The model owner decides when to make his simulation models available to a simulation and when to terminate their availability. The model owner interacts with the EODiSP through a model manager application.

The proposed operational concept is illustrated in figure 6 below for a distributed case. In the case illustrated in the figure, a simulation requires five models. Models A1 and A2 reside on a remote node (node 1 in the figure) and belong to model owner A. Model B1 resides on the same remote node but belongs to a different model owner B. Models C1 and C2 reside on the same node on which the simulation environment resides (node 2 in the figure) and belong to model owner C. In such a configuration, three model manager applications need to be active when the simulation manager application is started. In the figure, the applications are indicated as green boxes and the simulation models are indicated as yellow boxes.
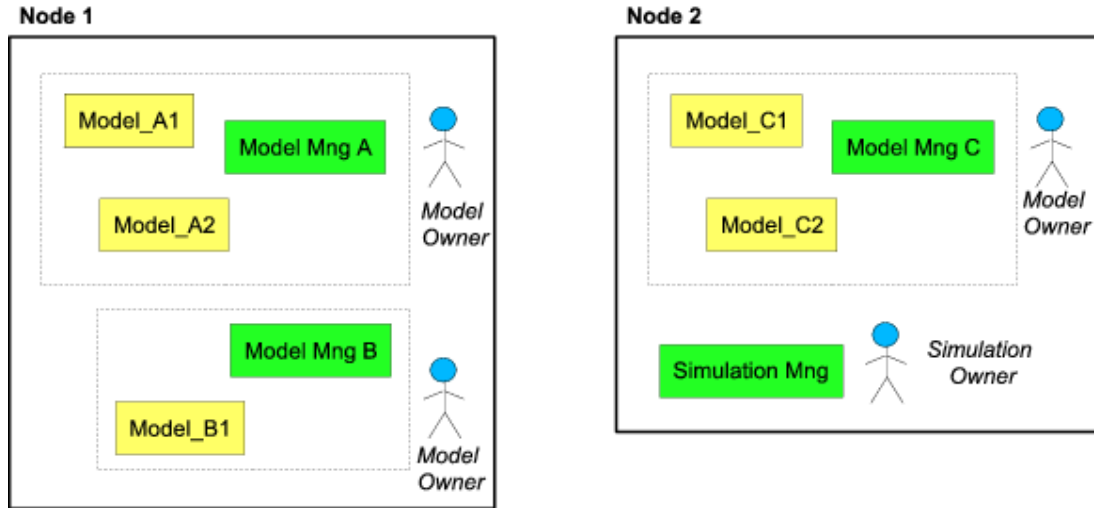

Fig. 6: EODiSP User Interface

EODISP REPOSITORY

The implementation of the concept outlined above requires the availability of a centralized repository where all models (suitably wrapped as HLA federates) that may take part in a simulation are registered. This repository is called the *EODiSP Repository*.

The registration of a model in the repository is done through the model manager application. The simulation manager application inspects the repository to check whether the models that are required for a given simulation are indeed available and to find information about their location.

The EODiSP can operate in two basic modes. In *remote mode*, one or more of the simulation models are located on a remote node (with respect to the simulation manager application). In *local mode*, all the simulation models are located on the same platform as the simulation manager application. Operation in remote mode requires the repository to be installed on a publicly accessible server. Operation in local mode requires the repository to be installed locally on the same machine on which the simulation models and the simulation manager application are located.

The EODiSP provides an application, the *repository manager application* to control the configuration and creation of an EODiSP repository.

TYPICAL USAGE SCENARIO

In the EODiSP concept, simulation models are encapsulated in simulation packages that exist independently of the simulation infrastructure. The first logical step towards the creation of a simulation is taken when model owners who are prepared to allow their simulation models to be used start the model manager GUI application on their node. They then register their models with the EODiSP repository. The information in the repository consists uniquely of a description of the external interfaces of the models. The model code remains under the complete control of the model owners. After the registration process is complete, the model manager remains in listening mode, waiting for a simulation manager application to request it to join in a simulation. The model owner can at any time withdraw his models from a simulation through the model manager GUI application. This mechanism thus ensures that model owners remain in control of their models. This is important in view of the fact that an EODiSP simulation may be made up of a collection of models that belong to different individuals or organisations that may be unwilling to hand over control over them.

A simulation owner controls a simulation through the simulation manager GUI application. He performs a simulation in three basic steps. In the first step, the simulation owner defines the simulation configuration by defining which simulation models participate in a simulation experiment, and how many and which simulation runs should be performed. The simulation owner has access to the EODiSP repository and can browse it to identify the models that are of interest to him. In the second step, the simulation owner starts the configured simulation experiment. In the third step, the simulation manager application identifies the model manager applications that give access to the simulation models required for the simulation and, if it finds them, starts execution of the simulation.

In principle, a simulation experiment, once it is started, will autonomously continue until all the simulation runs defined in the experiment have terminated. However, the EODiSP GUI also gives the simulation owner control over a running simulation. In particular, the simulation owner can ask for a running simulation to be held and it can ask for a simulation to execute in step-by-step mode. These facilities may be especially useful for debugging purposes.

SIMULATION LIFECYCLE

The lifecycle of an EODiSP simulation (or, more precisely, of an EODiSP HLA federation) is marked by five events:

1. EODISP_START: this event marks the start of a simulation. It is reached when all federates in a federation have completed their attribute publication and subscription process. After the EODiSP_START event has been reached, the simulation starts in the sense that the simulation federates begin to update the attributes they publish and begin to process notifications of updates to the attributes they subscribe to.
2. EODISP_PAUSE: this event changes the state of the simulation from 'running' to 'paused'. When a simulation is paused, the data flow between federates is temporarily suspended. A pause facility may be useful for debugging purposes or to allow simulation model maintenance during a simulation run. Pausing is implemented by building into one or more federates the ability to suspend the update of the attributes they publish.
3. EODISP_STEP: this event causes the execution of a paused simulation to advance of one step. The models that have suspended the update of the attributes they publish, release them but only once.
4. EODISP_RESUME: this event changes the state of the simulation from 'paused' to 'running'. The models that have suspended the update of the attributes they publish, release them.
5. EODISP_STOP: this event marks the end of a simulation. When a simulation stops, all federates cease to update the attributes they publish and resign from the federation.

The first four events occur in response to requests made by the simulation manager through the simulation manager application. The last event - the simulation stop - occurs as a result of the evolution of the internal state of one or more federates participating in a simulation.

The lifecycle events introduced above are managed through HLA synchronization points and HLA interactions. Synchronization points are a mechanism defined by the HLA to coordinates the actions of federates in a simulation. Generally speaking, synchronization points in HLA work as follows:

1. The control federate registers a synchronization point for a group of federates, called the synchronization group.
2. The RTI announces this newly registered synchronization point to all federates in the synchronization group.
3. Each federate of the synchronization group informs the RTI when it reaches the synchronization point (after receiving the synchronization point announcement).
4. The RTI informs all federates in the synchronization group that the synchronization point is achieved as soon as all federates have informed the RTI that they achieved it. At this point the federation is synchronized.
5. The synchronization point is removed from the RTI.

The EODiSP predefines five synchronization points, one for each of the five events described above. Additional synchronization points may of course be defined for a particular simulation. Federates announce their intention to be included in a synchronization group in the SOM.

Interactions are a mechanism defined by the HLA to allow a federate to perform an action or transfer some data to other federates in a federation. The EODiSP predefines one interaction. This is used in the initialization phase (i.e. before the EODiSP_START synchronization point has been achieved) to allow each federate to send a handle that uniquely identifes it to the control federate. Additional interactions may of course be defined for a particular simulation. Federates declare their interactions in the SOM.

EODISP DEMONSTRATOR

The EODiSP is provided with a demonstrator which is intended to verify its correct implementation and to serve as a blueprint of how a simulator can be set up as an instantiation of the EODiSP. The EODiSP demonstrator is based on the phase A simulator for the EarthCARE mission. It is described in [3].

IMPLEMENTATION

The whole EODiSP framework is implemented in Java. The graphical user interface is implemented in Java Swing. This choice guarantees portability on virtually any kind of desktop platform.

AVAILABILITY AND LICENCING SCHEME

The EODiSP is made available through a public web site [4] as free and open software under a GNU's General Public Licence (GPL)

REFERENCES

[1] Frederick Kuhl, Richard Weatherly, and Judith Dahmann, "Creating Computer Simulation Systems: An Introduction to the High Level Architecture", Prentice Hall, 1999
[2] http://standards.ieee.org/, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)", Documents 1516-2000, 1516.1-2000, 1516.2-2000, and 1516.3-2000
[3] http://www.pnp-software.com/earthcare_simulator/
[4] http://www.pnp-software.com/eodisp