Modeling with DEOS

A. Grieco, S. Zacchino, L. Castelluzzo, D. Colì
Dipartimento di Ingegneria dell'Innovazione - Università degli Studi di Lecce
via Arnesano 73100 Lecce - telefono: 0832 297251 - e-mail: antonio.grieco@unile.it

Contents

Chapter 1

The modeling approach

1.1 Mapping to DEOS model

The first step to create a simulation with DEOS is to represent systems in some object-oriented notation as UML, in order to describe them correctly and to make easier the model creation. At this stage, the most useful UML diagrams are:

- use case diagrams;
- activity diagrams.

Usually, they allow to find the system components, known as *simulation objects*, and reduce the risk of mapping the real system to a wrong model.

Next step is to state which components are eligible to be *resources* or to be *entities* according to DEOS model. DEOS was created to model manufacturing processes, but more generally it is effective to represent systems in which entities migrate between complex components called resources. Therefore, a component has to be mapped to a resource if its behaviour is to manipulate other components. Instead, if the object is simply manipulated by other components, it is mapped to an entity.

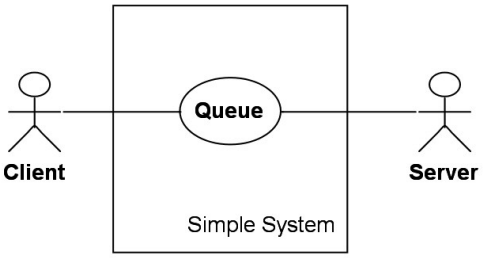

Figure 1.1: A simple queue system.

Figure 1.1 shows a simple use case diagram of a queue system where clients arrive and wait for the server to be idle. The diagram evidences the components of the system: some clients, a queue and a server. The server manifests a behavior by serving clients, so it will be mapped to a resource. The queue keeps clients until the server can accept another one, so it will be mapped to a resource too. The clients will be entities instead, because they are subject to server and queue actions.

Usually, the model needs some resources able to create the entities and to introduce them into the system. These resources, often called *creators*, do not always have corresponding components in the real world: they are only for the purpose of simulating the entry of entities into the system. In the same way, the model needs some resources able to remove entities from the system. These resources are called *destroyers* and are used to simulate the exit of entities. Other kinds of resources can be lacking in real world counterparts, such as the *switches* that direct entities towards specific directions.

The last step in planning a model is to draw the DEOS *environment*. This term refers to the set of all resource instances involved in the model and their connections. To perform this last task it's required to find all comunication needs between resources and to decide the behavior and the status of all simulation objects. The comunication include both information exchange and entity passing. At this stage, the most useful UML diagrams are:

- sequence diagrams;
- state diagrams;
- class diagrams.

The previous example can be mapped to the environment shown in figure 1.2. The environment contains a creator that generates entities with given interarrival times. These entities go to a queue and wait for server access. When the entities stop being served, they are removed by a destroyer from the environment.

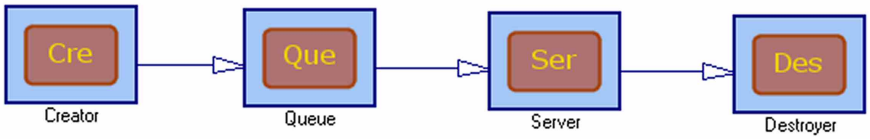

Figure 1.2: A simple environment.

When a model requires resources that do not already exist, a C++ or Tcl programmer must create DEOS plugins[1] representing these resources. He must give a form and a behavior to these resources following the model planning.

The model can be simplified by using object aggregation. In other words, several resources can be grouped in a single resource *group* which shows all properties and behaviors of the contained resources. In this way, a complex environment can be shown as if it includes a few macro-components instead of a lot of micro-components. Entity aggregation is quite different because entity groups don't exist, but the entities can be inserted in or extracted from any other entity and the container entity doesn't show the contained entity attributes. Entity aggregation is useful when resources must exchange many entities at a time.

1.2 The simulation objects

Each resource and each entity can keep information about the component it represents. A part or all of this information can be inspected from outside the simulation object. The visible part is

[1] see page 15

accessible in form of attributes. An attribute is a simulation variable or parameter associated to a simulation object and can be a double precision floating point number, a long integer or a string. Resource attributes representing parameters for the simulation are said *published* because the user can set them before starting the simulation.

DEOS allows to represent stochastic quantity through *random variables* associated to resources. Users can set the distributions of the random variables before simulation starts and, during the simulation execution, resources can get pseudo-random numbers with these distributions.

A resource can be equipped with input and output ports for communication purposes. Each port is associated to its description and has an identification number in the range $1, 2, \ldots$. During the environment composition the user can establish connections from an output port of a resource to an input port of another one.

As said above, new resources are formed by programmers: they can add attributes, random variables and I/O ports to a resource and make it use them. The user can only set already existing resource properties by using an object inspector as in figure 1.3. The example above didn't require to create new resources because the DEOS base resources fitted our needs.

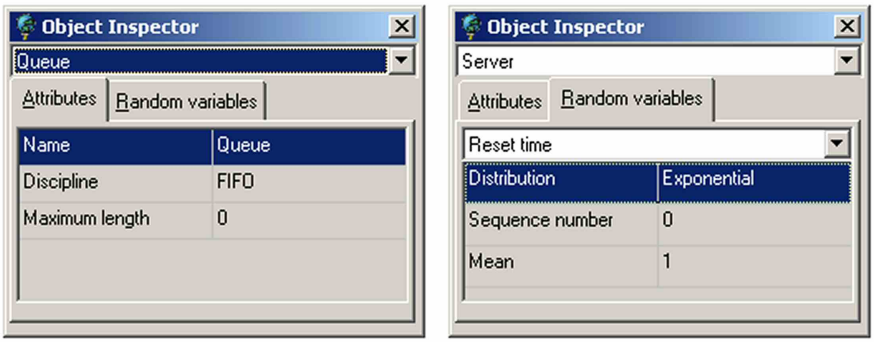

Figure 1.3: An object inspector.

1.3 Time and event management

A DEOS simulation can be viewed as a sequence of events. Each resource can ask the environment for scheduling an event in the present or in the future. The environment uses a timeline to collect scheduling requests and to order them by time. It keeps the current simulation time in a double precision floating-point variable which can be inspected by any resource. Every time an event execution ends, the timeline searches for the next event to execute (that is the one with the smallest execution time) and updates the current time variable. An environment is able to perform a specified number of simulation runs by resetting its timeline and all resources at each simulation end.

Programmers can associate *event types* to resources, besides the above properties. Each event type can have a handler method that allows the resources to have a behavior. A resource schedules only events belonging to its own event types and can disable future events, too. When the timeline has to execute an event of a given event type, it calls the handler method for that event type.

1.4 Resource communication

The connections between resource I/O ports are the channels through which resources exchange entities or information. A transfer protocol, called *WGN*[2], allows two resources to start and to complete an entity transfer. Both sending and receiving resource can take the initiative in starting a transfer. The WGN protocol involves the following method calls:

`Receiver.WorkRequest(InputNumber);` This method informs the receiver that the sender has an entity that is ready to be transferred. `InputNumber` keeps the input port identification number involved in the connection.

`Entity = Sender.GetOutput(OutputNumber);` The receiver tries to get the entity through this method. `OutputNumber` keeps the output port identification number involved in the connection.

`Sender.NextFree(OutputNumber);` The receiver can solicit the sender for preparing another entity transfer by this method. `OutputNumber` is used as in the previous method.

Although the typical scenario is that shown in figure 1.4, the protocol doesn't require to preserve the call order, but each resource must be able to handle every call order, even only with an error generation. This loose protocol definition allows to satisfy several comunication needs, but requires that communication partner resources comply with the expectations of each other.

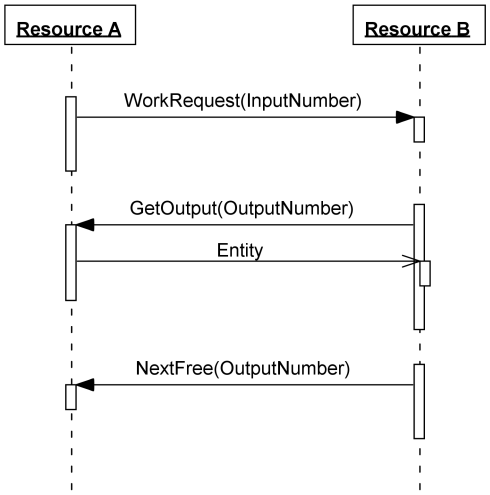

Figure 1.4: Typical entity transfer scenario.

Entity transfer is not the unique kind of communication between resources. Sometimes a resource has to get information from a partner resource attribute. There exist two methods that a resource can use for this purpose:

`Observed.GetCopyAttributeFromInput(InputNumber, Name, Copy);` This method write into `Copy` the content of the attribute with name `Name`. `InputNumber` keeps the input port identification number involved in the connection.

[2]WorkRequest, GetOutput and NextFree.

`Observed.GetCopyAttributeFromOutput(OutputNumber, Name, Copy);` The same syntax but `OutputNumber` keeps the output port identification number involved in the connection.

1.5 DEOS base resources

The *Base Plugins* library contains some common base resources that are general enough to be used in several contexts. These resources allow the user not to rewrite code for common tasks like creating, routing, enqueuing and destroying entities. They can also be used as base for writing new resources that share the same behaviour and properties. The list of the base resources is now shown. Near some attribute names there is the "NP" acronym which stands for "not published". It means that these attributes are not shown in the object inspector as parameters, but can only be used as simulation variables.

Branch Splits a flow of entities into two flows by using a probabilistic criterion.

Attributes:

- *Wait for chosen output*; if set to "Yes", when next output is chosen, but corresponding output resource is not free, entities will be not acquired. Otherwise, it doesn't wait for chosen output, but checks the other one and pushes next entity to it.
- *Entities to output 1* (NP); the number of entities sent to the output 1.
- *Entities to output 2* (NP); the number of entities sent to the output 2.

Random variables:

- *Go to output 1*; if computed value is not null, the output 1 will be chosen.

Event types:

- *Dispatch*; the moment when an entity passage happens from the input port to one of the output ports.

Input ports:

- *Input*;

Output ports:

- *Output 1*;
- *Output 2*;

Creator Creates a given number of entities with a given temporal distribution. If the next resource doesn't get last created entity at a creation event, creator discards it and creates a new one.

Attributes:

- *Create just on next free*; if set to "Yes", it only creates a new entity when *NextFree* method is called. Otherwise, it uses the random variable "Creation interval" to compute the creation times.
- *Create on start*; if set to "Yes", it creates a new entity on execution start, no matter what other properties hold.
- *Maximum of entities*; if null, any number of entities can be created. Otherwise, it limits this number.
- *Created entities* (NP); the number of entities created until now.

- *Discarded entities* (NP); the number of entities discarded until now.

Random variables:

- *Creation interval*; the time between two creations.

Event types:

- *Creation*; the moment when an entity creation occurs.
- *Discarding*; the moment when an entity discarding occurs.

Output ports:

- *Created entities exit*;

Destroyer Destroys entering entities and shows their lifetime.

Attributes:

- *Destroyed entities* (NP); the number of entities destroyed until now.
- *Last entity creation time* (NP); the creation time of the last destroyed entity.
- *Last entity lifetime* (NP); the lifetime of the last destroyed entity.

Event types:

- *Destruction*; the moment when an entity destruction occurs.

Input ports:

- *Destroyer entry*;

Queue Enqueues entities waiting for next resource acquisition. The user sets the maximum amount of entities, which may be infinite and the extraction policy, which may be FIFO or LIFO.

Attributes:

- *Discipline*; this attribute sets what kind of discipline the queue will obey. The two available values are: "FIFO" and "LIFO".
- *Maximum length*; if null, any number of entities can be holded in the queue. Otherwise, it sets the maximum for this number.
- *Number of entities* (NP); the number of the entities that are currently in the queue.

Event types:

- *Arrival to queue*; the moment when the queue gets a new entities from the input port.
- *Exit from queue*; the moment when the queue sends an entity to a free output resource.

Input ports:

- *Queue input*;

Output ports:

- *Queue output*;

Server Emulates a server by holding an entity until the service time is expired. It can wait for a reset time before starting a new service.

Attributes:

- *Free on next free*; if set to "Yes", the server waits for the call of the output resource method *NextFree* before calling the input resource method *NextFree*. In other words, it claims to be idle when the output resource does it.
- *Last reset time* (NP); the reset time after last service.
- *Last service time* (NP); self-explanatory.
- *Processed entities* (NP); the number of entities that have completed their service until now.

Random variables:

- *Reset time*; the time between the service end and the moment in which the server is ready to start another service. Sometimes it is called "setup time", too.
- *Service time*; the time between the service start and the service end.

Event types:

- *Begin service*; the moment when a service begins.
- *End service*; the moment when a service ends and a reset period starts.
- *End reset*; the moment when a reset period ends and the server is ready to get another entity.

Input ports:

- *Input*;

Output ports:

- *Output*;

Switch Gets entities from a given input and pushes them to a given output. It allows to specify the policy used to choose the input port and the output port of the next entity transfer.

Attributes:

- *Chosen input*; the number of the input port from which entities will be acquired.
- *Chosen output*; the number of the output port to which entities will pushed.
- *Input policy*; it sets the way the switch chooses the input port to get entities. This can be: "Only chosen input" to let "Chosen input" attribute decide, "Strict round robin" to get from the connected input ports in a strictly one by one way, "Loose round robin" to get from whatever input port and to serve concurrent requests with round robin algorithm, "With priority" to get from whatever input port and to serve concurrent requests with priority, where the lesser is the port number, the higher is its priority.
- *Output policy*; it sets the way the switch chooses the output port to push entities. This can be: "Only chosen output" to let "Chosen output" attribute decide, "Strict round robin" to push to the connected output ports in a strictly one by one way, "Loose round robin" to choose among the free output resources with round robin algorithm, "With priority" to choose among the free output resources with priority, where the lesser is the port number, the higher is its priority.
- *Last served input* (NP); the input port number involved in the last entity dispatch.
- *Last used output* (NP); the output port number involved in the last entity dispatch.

Event types:

- *Dispatch*; the moment when an entity dispatch happens.

Input ports:

- *Input 1...16*;

Output ports:

- *Output 1...16*;

1.6 Resource data acquisition

Resources show data to the whole environment by changing attribute values. The statistical data can be computed by collecting those values at given times. Therefore, the project of a new resource must take into account what variables the user will need. These variables should be kept inside attributes to allow the environment to access them. For instance, when a destroyer removes an entity from the environment, it writes the entity lifetime into the *Last entity lifetime* attribute, because it may be a useful information for statistical purpose.

Besides the input attributes, whose values are to be collected, the user is interested in the moment in which the collection is performed. This time could be:

- an attribute change;
- an event execution;
- a computation of a statistical value;
- a simulation run end.

In DEOS notation this time is referred to as *warner*. It must be specified by the user together with the attribute to collect. Attribute change is the most probable warner for attribute value collecting, so that it is the default choice.

Sometimes, the source of information the user needs is the warner itself. For instance, one may be interested in the simulation time of an event, in order to count its occurrences. Furthermore, the warner can be associated to a value that could be useful when the warner arises. In this case you can use a *trigger*, which is defined as a functional application $warner \rightarrow value$. For instance, to count the occurrences of an event, we have to set that event as the trigger warner and to let the trigger value equal 1. This value will be added everytime the event arises.

1.7 Data acquirers

The user can put plugin instances that aren't resources into environments, because they do not participate in the simulation but only look at the simulation data. These plugins are referred to as *data acquirers*. Their superclasses provide the programmers with interfaces and methods that ease the task of creating new data acquirer plugins.

Figure 1.5 shows all data acquirer types. The first level specializations are *monitor* and *gauge*. The latter is a plugin that employes acquired data to animate a graphical object during simulation execution. The user can specify the attributes to use for animating graphic objects. For instance, a progress bar like that in figure 1.6 is a simple gauge showing an input value as a bar length. Instead, monitors don't display any message during simulation runs, but use data for other purposes.

Monitors are further divided into two categories: stat collectors and controllers. Collecting data and computing statistical indexes are the tasks of the stat collectors. The user can set the following stat collector properties:

- reset data on simulation run end;
- the action that must be taken;
- the start time to begin collecting;

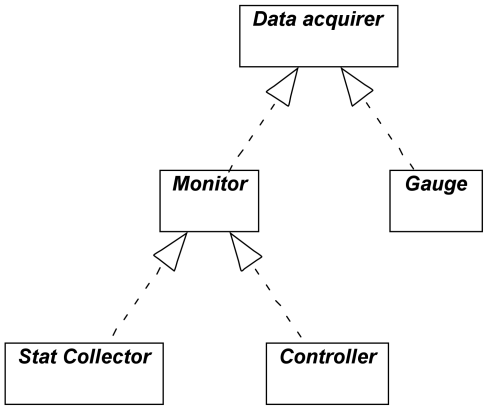

Figure 1.5: Data acquirer types.

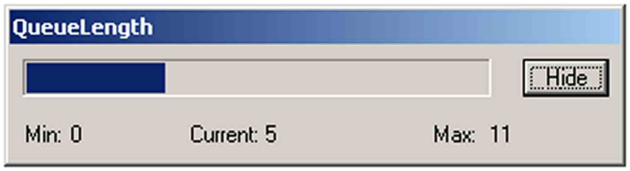

Figure 1.6: A progress bar showing a queue length.

- input attributes and warners;
- triggers.

Figure 1.7 shows the inspector that the user can employ to set the above properties.

Controllers are used to change resource attributes during the simulation execution when a condition becomes true. They allows the user to exert a feedback on the system, based on simulation variables. A controller needs to be set by its *targets*, which associate resource attributes to the values that should be taken when the condition arises. Furthermore, controllers should be made aware of the action to take and of some other properties depending on the action. Threshold is a typical controller that evaluates its targets when an attribute exceeds a value. Figure 1.8 shows the user interface to add target to a controller.

1.8 Base data acquirers

The list of the base data acquirers currently available in DEOS is now shown. Near the data acquirer names there are acronyms which stand for: "SC" stat collector, "C" controller and "G" gauge.

Collector (SC) collects attribute values and the times when they change. It's possible to look at the values on screen or to save them to a file.

Attributes:

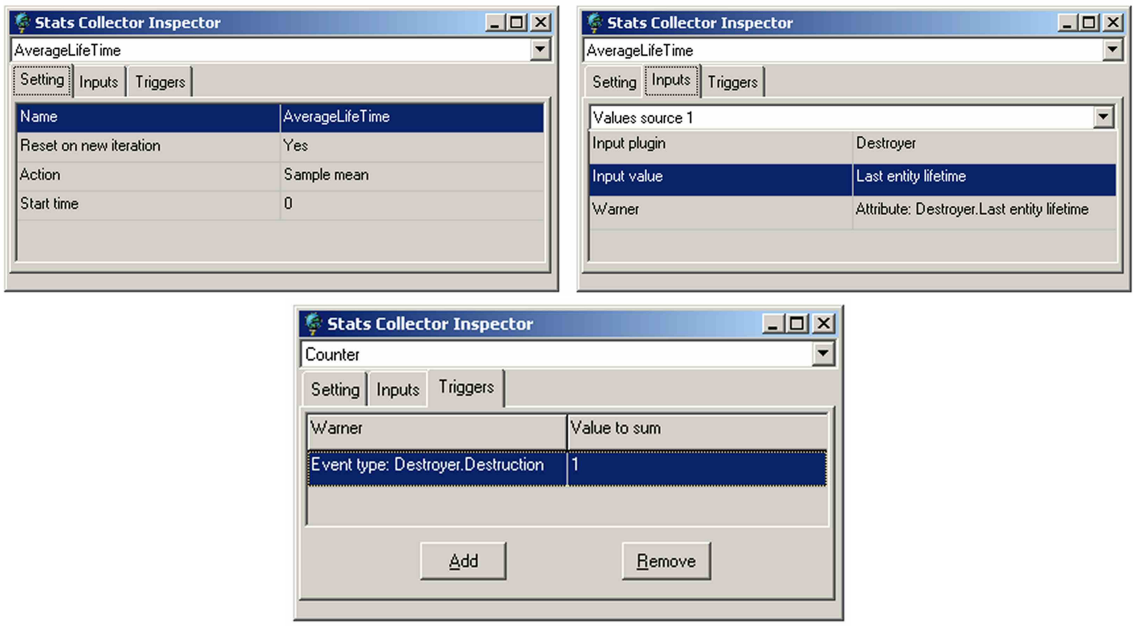

Figure 1.7: Stat collector inspector examples.

- *Reset on new iteration*; it allows to specify if collected data are deleted before starting every new simulation run.
- *Action*; if set to "Collect", it holds data into RAM. If set to "Save", it save data in a file.
- *Times*; it allows to decide what to get: only values, values and collection times, only collection times.
- *Initial value*; if set to "Collect it", a collection will be performed at start time.
- *File name*; the file path where data will be saved.

Inputs:

- *Value source*; the source of the data to get.

Counter (SC) adds a value specified by a trigger to a variable initialized to zero.

Attributes:

- *Reset on new iteration*; it allows to specify if the sum is set to zero before starting every new simulation run.
- *Action*; "Count" is the only available action.

Triggers:

- they allows to select the value to add for each specified warner;

Reporter (SC) reports information coming from inputs. It's possible to specify a template text file for the report.

Attributes:

- *Reset on new iteration*; useless.
- *Action*; "Report" is the only available action.

Figure 1.8: A controller inspector.

- *Template text file*; the ASCII file that contains the report format. The final report is written by replacing input tags with their corresponding values. Input tags are in the form: %F%N where F is the C printf style format for floating point numbers and N is the input number. For instance, %-7.0f%3 will be replaced with the input 3 value, left justified, in 7 characters padded with blanks and without decimals. Only numeric attributes can currently be used. If no template file is specified, the report will contain the current value of all connected inputs.

Inputs:

- *Value source 1...128*;

StatsMeasure (SC) performs statistical operation on attribute values. It can compute: mean, variance, moment, temporal average value, minimum and maximum.

Attributes:

- *Reset on new iteration*; if set to "Yes", all data are deleted before starting every new simulation run.
- *Action*; there is an action for each of the above computations.
- *Start time*; the time when data collection must begin.
- *Moment order*; it specifies the moment order when "action" is set to "Moment".

Inputs:

- *Value source*; the only availble input when computing temporal average value.
- *Value source 1...16*;

Expression (SC) computes numerical expressions that contain variables referring to the inputs, the simulation time and the current iteration number. It recognizes the most common mathematical operators and functions.

Attributes:

- *Reset on new iteration*; if set to "Yes", all data are deleted before starting every new simulation run.
- *Action*; "Evaluate" is the only available action.
- *Start time*; the time when data collection must begin.

- *Ignore warners*; if set to "Yes", the evaluation is not done when a warner arises, but when other data acquirers ask for the current value. Otherwise, the evaluation only happens when a warner arises.
- *Expression*; it is the expression to evaluate. Its format should be the C style format for numerical expressions except for it is case insensitive. It recognizes the following variables: i as the current iteration, $time$ as the current simulation time, vi as the value coming from the i-th input. The following operator can be used: $+$, $-$, $*$, $/$ and ^. Furthermore, it recognizes the following functions: sqr, log, ln, sin, cos, tan, $asin$, $acos$, $atan$, abs, int, $hsin$, $hcos$, $htan$ and $ispositive$. This last function returns 1 if the argument is ≥ 0.

Inputs:

- *Value source 1...32*;

Threshold (C) modifies a resource attribute when an input value matches a specific condition.

Attributes:

- *Action*; if set to "Assign", it will set the target attribute. If set to "Sum" it will add to the target attribute.
- *Condition*; the type of comparison between the input value and the comparing value.
- *Comparing value*; the right value of the comparison.

Inputs:

- *Value to check*; the source for the left values of the comparison.

Targets:

- They set the attribute to change and their target values.

ProgressBar (G) shows an attribute value as a bar length.

Attributes:

- *Minimum*; minimum value shown by the bar.
- *Maximum*; maximum value shown by the bar.
- *Delay*; after each graphical update the bar waits for a delay. This number is proportional to this delay.

Inputs:

- *Values source*;

Tracer (G) executes a simulation one event at a time.

Attributes:

- *Start time*; the start time to trace events.

Chapter 2

Creating new plugins

2.1 Plugins and plugin files

A *plugin* is a piece of code that extends the capabilities of a program. Namely, a DEOS plugin is like a class whose instances are employed by the user to build a simulation. For instance, the user can add some instances of the *Creator* plugin to an environment and sets their properties. A DEOS plugin may be:

- an environment;
- a resource;
- a data acquirer;
- a resource group.

At least three classes make a DEOS plugin:

Main class provides registry data like plugin name, instance local name, plugin type (resource, stat collector, etc,) and plugin category. Next two classes can be acquired through the main one. Usually, the name of this class begins with *TD*.

Logic class perfoms tasks related to the simulation. This class determines the resource properties and behavior or the data acquirer capabilities, etc. Usually, the name of this class begins with *TDL*.

Graphic class satisfies multimedia requirements. It provides an icon showing plugin instances on the environment and all of the forms the plugin needs. Usually, the name of this class begins with *TDG*.

A *plugin file* is an executable module[1] that contains a library of plugins. There aren't assumptions about the choice of the plugins to include in a plugin file. Each plugin file contains the code for all of the plugin classes and for a *proxy* class. The task of the proxy class is to provide information about the plugins that the file contains and to create new instances of them. The only symbol[2]

[1] A DLL in Windows and a shared object in Linux.

[2] The Windows/Borland version also needs to export the symbol *_SetApplication* for form management purposes.

that the plugin file must export is a function *_GetProxy* that returns a pointer to the plugin file proxy.

Objects coming from different plugins must be handled by each other, but often the class defining an object is only known inside the scope of its plugin file. Therefore, the objects can't access each other through their defining class but using *interfaces*, which are known in the scope of any plugin file. DEOS architecture uses pure abstract classes as interfaces and usually, interface names have the suffix *Interface*. When the objects of a class are to be handled outside the scope of the plugin file, this class must inherit an appropriate interface. For instance, the plugin file proxy class must implement the interface *TDProxyInterface*.

2.2 Classes to specialize

Usually, a DEOS plugin developer can write his classes by specializing the *plugin definition classes*. They are divided into *base classes* and interfaces. The base classes provide an API to access useful facilities and can be exploited to reuse code that implements common tasks. For instance, there are base classes implementing attributes, random variables and event types that fit the typical needs. Furthermore, there is no need to rewrite the code that adds attributes, random variables and event types to a resource everytime a new one is created, because all of these tasks can be inherited from *TDLResource*.

As said above, interfaces state which methods can be accessed by objects that are outside the scope of the plugin file. Each class playing a role in the DEOS architecture must comply with an interface. Base classes themselves often inherit interfaces, as *TDLResource* does by implementing *TDLResourceInterface*. DEOS plugin developers would better specialize base classes instead of interfaces, because in this way they'd take advantage of the already hard tested code. In fact, interfaces don't have any code, but only define methods. Tables 2.1 and 2.2 show the base classes that a developer can use and specialize for different purposes.

Plugin type	**Base class**	**Description**
Resource	TDResource	Inheritable by resource main classes
	TDLResource	Inheritable by resource logic classes.
	TDAttribute	Used for attributes.
	TDRandomVariable	Used for random variables.
	TDEventType	Used for event types.
	TDGResource	Inheritable by resource graphic classes.
Stat collector	TDStatsCollector	Inheritable by stat collector main classes
	TDLStatsCollector	Inheritable by stat collector logic classes.
	TDGStatsCollector	Inheritable by stat collector graphic classes.
Controller	TDController	Inheritable by controller main classes
	TDLController	Inheritable by controller logic classes.
	TDGController	Inheritable by controller graphic classes.
Gauge	TDGauge	Inheritable by gauge main classes
	TDLGauge	Inheritable by gauge logic classes.

Plugin type	Base class	Description
	TDGGauge	Inheritable by gauge graphic classes.
Group	TDGroup	Used for the *Standard Group* main class.
	TDLGroup	Used for the *Standard Group* logic class.
	TDGGroup	Used for the *Standard Group* graphic class.
Environment	TDEnvironment	Used for the *Standard Environment* main class.
	TDLEnvironment	Used for the *Standard Environment* logic class.
	TDEntity	Used for entities.
	TDTimeline	Used for the *Standard Environment* timeline.
	TDPriorityQueue	Locally used for the event priority queue of the *Standard Environment* timeline.
	TDSimulationEvent	Used for the *Standard Environment* events.
	TDGEnvironment	Used for the *Standard Environment* graphic class.
	Others	Platform dependent locally used classes for multimedia purposes.

Table 2.1: Base classes useful for specific plugin types.

Base class	Description
TDeosException	Used for throwing exceptions caused by blocking errors during simulation execution.
TDProxy	Inheritable by proxy classes. The task to give information about the plugins and to instance them gets easier by inheriting this class.
DHook	Template used for holding and passing pointers to class methods that have the form `void Method(TDHookInterface *)`, where *TDHookInterface* is the interface implemented by *DHook* itself. In DEOS notation, a hook is an instance of this template.
DVector	Template used for holding and passing pointer vectors. The pointers are in the form `T *` where *T* is the template defining class.
TDStrings	Used for holding and passing string vectors.

Table 2.2: Base classes useful in various cases.

For a complete treatment of the base classes, refer to the "DEOS API documentation". However, let's analyse some of them now.

TDLResource This abstract class provides facilities for the plugin developer like adding, accessing and deleting attributes, random variables, event types, input ports and output ports. All the data managed by this class is saved or loaded by its XML methods. The developer must always override the pure virtual methods: *WorkRequest* and *NextFree* when deriving a class from this one. Furthermore, WGN protocol methods of partner classes can be called by simply using internal methods designed for this. For instance, to call the *WorkRequest* method of the resource connected to output 2, the right call is: *WorkRequestToOutput(2)*.

This class provides the resources with features that make them cooperates with other objects. For instance, the resources are able to be grouped by a group plugin and to set published attributes to their initial values when a simulation run starts.

TDAttribute This class implements an attribute. It can be viewed as a variable that has a name, a type, a content and a range of correct values. When it belongs to a resource, it can be published or not. The types currently supported are: "String", "Integer" and "Double". The attribute interface allows the developer to set everyone of these properties.

Furthemore, the developer can add warnings to an attribute. A warning is a hook to a class method that the attribute calls when its content changes. This is how the data acquirer warners work.

TDRandomVariable Class for random variables. The properties of a random variable are: a sequence number, a distribution and related data. The sequence number can be an integer in $[0, 15]$ and identifies which of the uniformly distributed pseudo-random number sequence, generated by the environment, must be used.

The distribution currently supported are:

Fixed a constant *Value* with probability 1.

Uniform uniformly distributed in [*LowerBound*, *UpperBound*].

Normal normal with given *Mean* and *Standard deviation*.

Exponential negative exponential with given *Mean*.

Triangular triangular with given *Min*imum, *Mode* and *Max*imum.

Binary 1 with probability *Success probability*, otherwise 0.

Discrete discrete with a given *Distribution data*, which is a sequence of numbers $v_1\ p_1\ v_2\ p_2\ \ldots\ v_n\ p_n$, where v_i occurs with probability $p_i / \sum p_i$.

TDEventType Resources have an object of this class for each type of event they can schedule. An event type can have: a name and an event function (also known as handler). This class provides the developer with methods that schedule and disable events, add and remove warnings and return information about events. Every scheduled event is associated with an identification number through which the event can be referred.

TDEntity Resources can ask the environemnt for generating entities. The standard environment instances them by using this class. Its methods allow to add, modify and remove attributes, and to set aggregation relationship between entities.

TDeosException During a simulation run, if an unrecoverable error occurs, an exception should be thrown. The standard environment catch exceptions of this class.

2.3 Creating resources

The most common case of plugin implementing is for creating new resources. This section shows the steps to write the code for a new resource by using all of the facilities that base classes allow. The reader should have a good C++ knowledge to understand the example code. Furthermore, we assume that the resource has already been designed in terms of behaviour, attributes, random variables and event types[3]. Currently, graphic classes have only been written for the Windows/Borland platform, so our graphic class example is only valid in this context.

2.3.1 Source files

Let's suppose we have to create a resource named "Foo", which will be contained in a plugin file named "MyPluginFile.dll". The base directory tree we need is that shown in figure 2.1, but in the real case other directories may exist. The grayed zone is the one the developer must create, whereas the rest of the directories should already exist. By convention, *MyPluginFile* directory is

[3]See page 3.

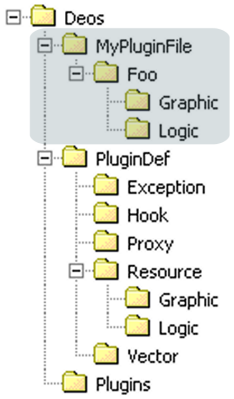

Figure 2.1: The directory tree to create the Foo plugin.

at the same level of *PluginDef*, which contains the plugin definition classes and of *Plugins*, which is the destination of the plugin file. Table 2.3 shows what the *MyPluginFile* and its subdirectories contain. These files will be better explained below.

Directory	**File**	**Content**
MyPluginFile	MyPluginFileExport.cpp	The symbols exported by the plugin file (*_GetProxy* and *_SetApplication*).
	ProxyMyPluginFile.h	Declaration of the proxy class *TDProxyMyPluginFile*.
	ProxyMyPluginFile.cpp	Code for the proxy class. This file contains a description of the "Foo" resource and the code to instance it.
	MyPluginFile.bpr	C++ Builder project file.
Foo	DFoo.h	Declaration of the main class *TDFoo*.
	DFoo.cpp	Code for *TDFoo*. Usually, this code assigns a name to the plugin and instances the logic and graphic classes.
Graphic	DGFoo.h	Declaration of the graphic class *TDGFoo*.
	DGFoo.cpp	Code for *TDGFoo*. It provides the icon to represent the plugin instances and eventually a window for specific settings.
	DGFoo.rc	The definition for the bitmap file of the icon as DLL resource.
	DFoo.bmp	The bitmap file of the icon. Usually (80x60 pixels with 8 bit per pixel).
Logic	DLFoo.h	Declaration of the logic class *TDLFoo*.
	DLFoo.cpp	Code for *TDLFoo*. The behaviour and all of the other features related to the model are implemented here.

Table 2.3: *MyPluginFile* and its subdirectories content.

To create a DLL project with C++ Builder you must select *File* → *New* → *Other* and choose *DLL Wizard* from the *New* tab. Now, check *C++* as source language and *Use VCL*. Save the unit as *MyPluginFileExport.cpp* and the project as *MyPluginFile.bpr*. After this, select *Project* → *Options*, go to the *Directories/Conditionals* tab and set *Final output* to "..\Plugins" to make the DLL available to DEOS.

We advice to modify the following options: select *Project → Options* and to uncheck *Use dynamic RTL* from the *Linker* tab and *Build with runtime packages* from the *Packages* tab. Sometimes it's necessary to add manually some static library to the project file: select *Project → Edit Option Source* and let the *LIBRARIES* and *SPARELIBS* elements have an attribute *value="rtl.lib vclx.lib vcl.lib"*.

Furthermore, the linker needs some *.lib* and *.obj* files that are obtained by compiling the base classes. Table 2.4 shows the complete list of these files. The files to include into the project depend on the type of the new plugins.

Plugin type	**File**
Resource	Deos/PluginDef/Resource/Resource.lib
Stat collector	Deos/PluginDef/StatsCollector/StatsCollector.lib
Controller	Deos/PluginDef/Controller/Controller.lib
Gauge	Deos/PluginDef/Gauge/Gauge.lib
Group	Deos/PluginDef/Group/Group.lib
Environment	Deos/PluginDef/Environment/Environment.lib
Any	Deos/PluginDef/Exception/Exception.lib
Any	Deos/PluginDef/Proxy/Proxy.lib
Any	Deos/PluginDef/Vector/DStrings.obj

Table 2.4: Compiled files to include for different plugin types

2.3.2 Plugin file code

Let's see the code for exporting symbols.

```
1  // MyPluginFileExport.cpp
2  #include <vcl.h>
3  #include <windows.h>
4  #pragma hdrstop
5  #include "ProxyMyPluginFile.h"
6
7  extern "C" __declspec(dllexport) TDProxyInterface *GetProxy()
8  {
9      return new TDProxyMyPluginFile();
10 }
11
12 extern "C" __declspec(dllexport) void SetApplication(void *MainApp)
13 {
14     static TApplication *MainApplication = Application;
15
16     if (MainApp) Application = (TApplication *) MainApp;
17     else Application = MainApplication;
18 }
```

Lines 2 and 3 include the Visual Component Library and the Windows header files. Line 4 tells the C++ Builder preprocessor to stop precompiling headers. Line 5 includes the proxy class header file. The function definitions in lines 7 and 12 use the modifiers ***extern*** *"C" __declspec (dllexport)* to tell the compiler that following symbols must be exported. Line 9 instances a proxy and returns it to the caller. Lines 12-18 should always be the same, because they simply make the DLL instance use the *TApplication* object of the main application.

The code for the proxy declaration and definition is shown now.

```
1  // ProxyMyPluginFile.h
2  #ifndef ProxyMyPluginFileH
3  #define ProxyMyPluginFileH
4
5  #include "Foo/DFoo.h"
6  #include "../PluginDef/Proxy/DProxy.h"
7
8  class TDProxyMyPluginFile : public TDProxy
9  {
10     public:
11         TDProxyMyPluginFile();
12         virtual TDPluginInterface *NewPluginInstance(const char *PluginName);
13 };
14
15 #endif
```

```
16 // ProxyMyPluginFile.cpp
17 #include "ProxyMyPluginFile.h"
18
19 TDProxyMyPluginFile::TDProxyMyPluginFile()
20 {
21     AddPluginInfo("Foo", "It's a test resource.",
22         "Resource", DGeneral);
23 }
24
25 TDPluginInterface *TDProxyMyPluginFile::NewPluginInstance(const char *PluginName)
26 {
27     if ( string(PluginName) == "Foo") return new TDFoo();
28     return 0;
29 }
```

Lines 2 and 3 avoid multiple declarations. Lines 5 and 6 include the declaration of the *TDFoo* class and of the base class *TDProxy*, respectively. Lines 8-13 declare the new proxy class peculiar to our plugin file. Lines 19-23 define the proxy constructor which adds the following properties about the plugin "Foo" to the proxy: name, description, plugin type (resource) and plugin category. The last property is usually set to *DGeneral* and will be used for future extended plugins which will be incompatible with the standard environment. Lines 25-29 define the *NewPluginInstance* method of the proxy, which instances one of the plugins it contains.

2.3.3 Main and graphic classes

Usually, writing main and graphic classes of a resource is quite easy. They only require a few lines of code to set the plugin name and icon. Let's see the main class code of the "Foo" resource.

```
1  // DFoo.h
2  #ifndef DFooH
3  #define DFooH
4
5  #include "Logic/DLFoo.h"
6  #include "Graphic/DGFoo.h"
7  #include "../PluginDef/Resource/DResource.h"
```

```
8
9  class  TDFoo : public TDResource
10 {
11     public:
12         TDFoo();
13         virtual  void  Initialize  ();
14 };
15
16 #endif
```

```
17 //  DFoo.cpp
18 #include "DFoo.h"
19
20 TDFoo::TDFoo()
21 {
22     MyName = "Foo";
23 }
24
25 void TDFoo::Initialize ()
26 {
27     if  (! MyLogic) MyLogic = new TDLFoo(this);
28     if  (! MyGraphic) MyGraphic = new TDGFoo(this);
29 }
```

Lines 5 and 6 include logic and graphic class declarations. Line 7 includes the declaration of *TDResource* which should be a base class for any resource main class. Lines 9-14 declare the main class *TDFoo* and the constructor line 22 sets the resource name. Lines 25-29 define the *Initialize* method, which instances the graphic and the logic classes. This method is usually called by the environment soon after the main class instantiation.

Now, let's see the code of the graphic class *TDGFoo*.

```
1  //  DGFoo.h
2  #ifndef DGFooH
3  #define DGFooH
4
5  #include "../../ PluginDef/Resource/Graphic/DGResource.h"
6
7  class  TDGFoo : public TDGResource
8  {
9      public:
10         TDGFoo(TDPluginInterface *Plugin);
11         virtual  HBITMAP GetBitmap();
12         virtual  void ShowPersonalForm();
13 };
14
15 #endif
```

```
17 //  DGFoo.cpp
18 #include "DGFoo.h"
19
20 TDGFoo::TDGFoo(TDPluginInterface *Plugin) : TDGResource(Plugin)
21 {}
22
23 HBITMAP TDGFoo::GetBitmap()
```

```
24 {
25     if (! MyImage) return 0;
26     MyImage->Bitmap->LoadFromResourceName((unsigned int) HInstance, "FOOBMP");
27     return MyImage->Bitmap->Handle;
28 }
29
30 void TDGFoo::ShowPersonalForm()
31 {}
```

Line 5 includes the declaration of TDGResource, which should be the base class for any resource graphic class. Lines 7-13 declare the graphic class *TDGFoo*. The constructor at lines 20-21 doesn't take any action, and this is the common case. The *GetBitmap* method at lines 23-28 tries to load a bitmap into the *TPicture* object pointed by *MyImage* and to return the image handle. The string *"FOOBMP"* identifies a DLL resource defined in the resource file *DGFoo.rc*. This file, which should be included into the project, has the following content:

```
FOOBMP BITMAP DISCARDABLE "DFoo.bmp"
```

DFoo.bmp is the bitmap file holding the plugin icon.

Last *DGFoo.cpp* method *ShowPersonalForm* could be used to open a window called *plugin dialog*. Usually, resources don't need such a feature but this is useful for other plugin types. For instance, a stat collector can show its result by opening this window.

2.3.4 Logic class

The greatest effort is usually made to write the logic class, because it describes the plugin properties and above all, it implements the resource behaviour. The developer can save a lot of lines of code by inheriting the base class *TDLResource*, which offers methods for common tasks of resources.

Let's suppose our resource "Foo" has this behaviour:

- it receives an entity from an input port;
- it adds an attribute holding a pseudo-random number to the entity;
- it tries to give out this entity through an output;
- if no resource acquires this entity, it stops receiving other ones.

Furthermore, let's suppose "Foo" has the following properties:

Processed entities attribute that holds the number of processed entities.

Label value random variable that generates the numbers to put into the entity attributes.

Entity arrival event type that arises when "Foo" gets an entity.

Let's see the declaration of the logic class of "Foo".

```
1 // DLFoo.h
2 #ifndef DLFooH
3 #define DLFooH
4
```

```
5  #include "../../PluginDef/Resource/Logic/DLResource.h"
6
7  class TDLFoo : public TDLResource
8  {
9      protected:
10         // Rapid accesses
11         TDAttribute *ProcessedEntities;
12         TDRandomVariable *LabelValue;
13         TDEventType *EntityArrival;
14
15         //Other Variables
16         bool SuspendedRequest, Ready;
17
18         // Event handlers
19         virtual void EntityArrivalHandler(TDHookInterface *AWarning);
20
21     public:
22         TDLFoo(TDPluginInterface *Plugin);
23         ~TDLFoo();
24         virtual void DeleteYourself();
25
26         virtual void WorkRequest(unsigned int InputNumber);
27         virtual void NextFree(unsigned int OutputNumber);
28         virtual void Reset();
29
30         virtual bool GetXmlStatus(TDXmlWriterInterface *AWriter);
31         virtual bool SetXmlStatus(TDXmlReaderInterface *AReader);
32 };
33
34 #endif
```

Lines 7-32 declare the logic class *TDLFoo*. Lines 11, 12 and 13 declare pointers for the properties that will be added. It's not necessary to preserve these pointers because *TDLResource* does it, but doing this we can access the properties faster. Line 16 declares two boolean flags which are used into the code: *Ready* is set to false everytime "Foo" receives a *WorkRequest* and decides to get next entity. It becomes true when the output resource call the *NextFree* method; *SuspendedRequest* is set to true everytime a *WorkRequest* call occurs and "Foo" is not able to process another entity. When the *NextFree* method runs, it will check if there are suspended requests to process. *EntityArrivalHandler* is the handler method for the event type "Entity arrival".

Let's see the code for the logic class of "Foo".

```
35 // DLFoo.cpp
36 #include "DLFoo.h"
37
38 TDLFoo::TDLFoo(TDPluginInterface *Plugin) : TDLResource(Plugin)
39 {
40     // Attributes ********************
41     *(ProcessedEntities = AddAttribute("Processed_entities")) = 0L;
42     ProcessedEntities->SetPublished(false);
43
44     // Random Variables ***************
45     (LabelValue = AddRandomVariable("Label_value"))->SetDistribution("Fixed");
46
```

```
47      // Event Types ********************
48      (EntityArrival = AddEventType("Entity arrival"))->
49          SetEventFunction(new DHook<TDLFoo> (this, &TDLFoo::EntityArrivalHandler));
50
51      // Inputs ************************
52      AddInput("Input");
53
54      // Outputs ************************
55      AddOutput("Output");
56
57      // Initializations
58      SuspendedRequest = false;
59      Ready = true;
60  }
```

Lines 38-60 define the constructor. Line 41 adds an attribute named "Processed entities" and sets it to 0. Line 42 makes the above attribute not published, that is it will be not shown in the object inspector among the resource parameters. In fact, it is a simulation variable and is not a parameter. Line 45 adds a random variable named "Label value" and sets its default distribution to "Fixed". Lines 48 and 49 add an event type and link it to its handler method. Lines 52 and 55 add an input port named "Input" and an output port named "Output", respectively. The enumerations of the input ports and of the output ports start from 1 and increment by one at every adding, so the "Input" port will be assigned to 1 and the "Output" port will be assigned to 1. Lines 58 and 59 initialize the flags.

```
61  TDLFoo::~TDLFoo()
62  {}
63
64  void TDLFoo::DeleteYourself()
65  {
66      delete this;
67  }
```

To delete an object that was instanced into another executable module, it's not possible to use the *delete* operator, because different modules have different memory managers. We have to invoke the self deletion of the object, instead. This is achieved by calling the *DeleteYourself* method which can safely do it.

```
68  void TDLFoo::WorkRequest(unsigned int InputNumber)
69  {
70      if (InputNumber == 0) return;
71      else if (InputNumber == 1)
72      {
73          if (! Ready || OutputEntities[0])
74          {
75            SuspendedRequest = true;
76            return;
77          }
78          EntityArrival->ScheduleIn(0.0);
79          Ready = false;
80      }
81      else throw TDeosException("Foo error: WorkRequest to a wrong input number");
82  }
```

Lines 68-82 define the *WorkRequest* method. Line 70 checks if the choosen *InputNumber* is 0. This particular value is used by the environment to call the *WorkRequest* methods of all the resources and to tell them that the simulation is starting. If *InputNumber* is equal to 1, line 73 checks if the resource is ready to process another entity. This line looks at the flag *Ready* and checks the output buffer, too. This prevents the output resource from causing a crash by calling *NextFree* before getting the entity. If the resource isn't able to process another entity, it sets *SuspendRequest* to true and returns (lines 75 and 76). Otherwise it schedules an "Entity arrival" event at this moment and sets *Ready* to false (lines 78 and 79). If the *InputNumber* value is incorrect line 81 arises a simulation error and stops the execution.

Note that the output buffer *OutputEntities* is defined in *TDLResource* as a vector of *TDEntityInterface* pointers and that its i-th item holds the pointer to the entity directed to the output $i + 1$.

```
83 void TDLFoo::EntityArrivalHandler(TDHookInterface *AWarning)
84 {
85     TDAttributeInterface *aLabel;
86     double value;
87
88     if (OutputEntities[0])
89         throw TDeosException("Foo error: output buffer contains an unexpected entity");
90     OutputEntities[0] = GetEntityFromInput(1);
91     if (!OutputEntities[0])
92         throw TDeosException("Foo error: impossible to get an entity from the input");
93     OutputEntities[0]->AddAttribute("Label");
94     aLabel = OutputEntities[0]->GetAttribute("Label");
95     if (!aLabel)
96         throw TDeosException("Foo error: impossible to add an attribute to the entity");
97     LabelValue->ComputeValue(value);
98     *aLabel = value;
99     *ProcessedEntities = (long) *ProcessedEntities + 1L;
100    WorkRequestToOutput(1);
101 }
```

Lines 83-101 define the handler method for the "Entity arrival" event. Note that the *AWarning* parameter of this method holds the pointer to the hook that was assigned to the "Entity arrival" event type. If a method is the handler of more than one event type, it can know what event occurs by reading the hook pointer.

Lines 88 checks if the entity buffer is empty and if not, then line 89 arises a simulation error. It should be empty because otherwise this event wouldn't have been scheduled. However, checking the output buffer before writing it, is a safe way to prevent entities from disappearing. Line 90 tries to acquire an entity from input port 1. The method *GetEntityFromInput* is a *TDLResource* facility to call the *GetOutput* method of an input resource. Line 91 checks if an entity was actually got and if not, then line 92 arises a simulation error.

Line 93 tries to add an attribute named "Label" to the acquired entity. The *AddAttribute* method returns the pointer to the new attribute if it was added. Instead, if an attribute with this name already exists or if an error occurred, it returns a null pointer. To check if the attribute actually exists, line 94 tries to acquire it through the *GetAttribute* method. If this attempt was unsuccessful, line 96 arises an error.

Line 97 generates a pseudo-random number by using the *ComputeValue* method of the random variable pointed by *LabelValue*. Line 98 lets the attribute "Label" be equal to this number. Line 99 increments the attribute "Processed entities" by one as long integer value. Line 100 tells the re-

source connected to the output port 1 that a new entity can be acquired. The *WorkRequestToOutput* method is a *TDLResource* facility to call the *WorkRequest* method of an output resource.

```
102 void TDLFoo::NextFree(unsigned int OutputNumber)
103 {
104     Ready = true;
105     if (SuspendedRequest) WorkRequest(1);
106 }
107
108 void TDLFoo::Reset()
109 {
110     TDLResource::Reset();
111     *ProcessedEntities = 0L;
112     SuspendedRequest = false;
113     Ready = true;
114 }
```

Lines 102-106 define the *NextFree* method which is called by an output resource that claims its availability. Line 104 sets the *Ready* flag to true to allow other entity acquisitions. Line 105 checks if there has been a suspended request. The way to grant the request is by calling its own *WorkRequest* method to pretend a request from outside.

Lines 108-114 define the *Reset* method which is called by the environment to reset the resource status. Line 110 calls *TDLResource* version of the *Reset* method. This is always necessary to allow resetting of all things managed by *TDLResource*. These things involve: deleting pending events and emptying the output entity buffer. Next three lines set the flags and the attribute "Processed entity" to the initial values.

This is the typical way to implement a resource that acts as a server. At line 78 in the *WorkRequest* method, one can specify a non-null time to simulate a service time. This time could be computed through a random variable to simulate the service randomness.

2.3.5 XML representation

A careful reader should be aware of a lack in the previous subsection: no definition for the XML methods was shown. The "Foo" resource could actually work without any XML methods during a simulation execution, but all tasks related to saving its status couldn't be safely made. In fact, the environment, which usually call this methods, couldn't get and set the whole status of the resource.

The XML methods, usually overridden, are *GetXmlStatus* and *SetXmlStatus*. The former is called by the environment to get the status of the resource. The status has to be written by means of a *TDXmlWriter* object, which is able to translate data to XML. The latter is called by the environment to set the resource status to that holded by a *TDXmlReader* object. This object allows the resource to browse into the XML description of its status.

Both methods return true on success and false on failure. Let's see the code for the "Foo" XML methods.

```
115 bool TDLFoo::GetXmlStatus(TDXmlWriterInterface *AWriter)
116 {
117     if (! TDLResource::GetXmlStatus(AWriter)) return false;
118     if (! AWriter->AddElement(DFLAGS)) return false;
119     AWriter->AddContent(SuspendedRequest);
```

```
120     AWriter->AddContent(Ready);
121     return true;
122 }
```

Lines 115-122 define the method *GetXmlStatus*. Line 117 calls the *TDLResource* version of this method to allow it to write attributes and random variables. In fact, only new members declared by the user should be written in the overridden methods. We only have to write the two flags *SuspendedRequest* and *Ready*. Before doing this, line 118 tries to add an XML element that will hold the two flags. *DFLAGS* is a constant holding the element name which is defined in "XmlElementNames.h". Lines 119 and 120 add the flags to the XML element separating them by a blank character. The method *AddContent* can add integer and floating point numbers, boolean values and strings. Note that it is not prudent to add more than one string into a single XML element because the string boundaries will be not preserved.

```
123 bool TDLFoo::SetXmlStatus(TDXmlReaderInterface *AReader)
124 {
125     if (! TDLResource::SetXmlStatus(AReader)) return false;
126     if ( string(AReader->GetElementName()) != DFLAGS) return false;
127     AReader->GetItem(SuspendedRequest);
128     AReader->GetItem(Ready);
129     AReader->GoNext();
130     return true;
131 }
```

Lines 123-131 define the method *SetXmlStatus*. Line 125 allows *TDLResource* to set self-managed properties to the data holded by the *TDXmlReader* object pointed by *AReader*. When *TDLResource::SetXmlStatus* returns, the object should be positioned on the first element after the last one read. Line 126 checks the name of the current element. Lines 127 and 128 read the flags. *GetItem* can read the same data *AddContent* can add, but in the string case, it only reads until a control character (blank, carriage return, etc.) is met. To get the whole element content as a string, *GetText* should be used instead. Line 129 moves the *TDXmlReader* object to the next element, if one exists.

Sometimes the data to write is more complex and requires browsing nested XML elements. Both the XML reader and the XML writer provide method to do it. Table 2.5 shows these methods and explains their use for both objects.

Method	**Writer use**	**Reader use**
Descend	It is used before putting nested XML elements into the current element. It makes the writer descend to a deeper nesting level before using the method *AddElement*.	It tries to descend to a deeper level than that of the current XML element. If at least a nested element exists, it moves the object on it and returns true.
Ascend	When all the nested elements of an element are written, this method allows to return to the higher level. Although allowed by XML, the capability to add element content after nested elements was not implemented. Therefore, after going back from nested elements it is only possible to add another element or to ascend again.	After reading all the nested elements of an element, this method allows to return to the higher level. The same notes of the writer hold.

Method	Writer use	Reader use
GoNext	Not used.	It is used to move the reader to the next element at the same level. If no more element exists it returns false.

Table 2.5: XML browsing methods.

When the data to write contains references to other simulation objects, *GetXmlReferences* and *SetXmlReferences* should be used instead of *GetXmlStatus* and *SetXmlStatus*. In fact, they are called by the environment when all of its plugins have already been instanced. This insures that all references can be resolved. For instance, the following piece of code adds an entity to the object pointed by *AWriter*. Since an entity could contain references, it's safer to put this code into the method *GetXmlReferences*.

```
1  if (! AWriter->Descend()) return false;
2  if ( AnEntity)
3      if (! AnEntity->GetXmlStatus(AWriter)) return false;
4  if (! AWriter->Ascend()) return false;
```

Note that lines 1 and 4 make the entity to add all of its XML elements as nested elements of the current one. If these lines didn't exist, the entity would have added all its elements after the current one. Line 2 checks if the entity exists, and if not, it doesn't write anything. The following piece of code is able to read the data written with the previous one.

```
1  if ( AReader->Descend())
2  {
3      AnEntity = Environment->GetNewEntity(0.0, 0L, "");
4      if (! AnEntity->SetXmlStatus(AReader)) return false;
5      if (! AReader->Ascend()) return false;
6  }
7  if (! AReader->GoNext()) return false;
```

Before reading data, line 1 checks if there are nested elements. Line 3 tells the environment to instance a new entity and puts its pointer into *AnEntity*. Line 4 makes the entity set its status to that specified by *AReader*. Line 5 returns to the wrapping element and finally, line 7 tries to go to the next element. In this case a next element is expected, so if it doesn't exist, this method returns false.

Chapter 3

DEOS License

3.1 Adopted licenses

The Mozilla Public License Version 1.1 (MPL 1.1) applies to all of the files in Deos directory and recursively in all subdirectories.

Furthermore, the files in the following subdirectories:

- PluginDef/
- PluginDef/Resource/
- PluginDef/Resource/Logic/
- PluginDef/Resource/Graphic/
- PluginDef/StatsCollector/
- PluginDef/StatsCollector/Logic/
- PluginDef/StatsCollector/Graphic/
- PluginDef/StatsCollector/Graphic/Plot
- PluginDef/Controller/
- PluginDef/Controller/Logic/
- PluginDef/Controller/Graphic/
- PluginDef/Gauge/
- PluginDef/Gauge/Logic/
- PluginDef/Gauge/Graphic/
- PluginDef/Exception/
- PluginDef/Hook/
- PluginDef/Plugin/

- PluginDef/Proxy/
- PluginDef/Misc/
- PluginDef/Vector/

are also licensed under the terms of the Academic Public License Vesion 2.1 (AFL 2.1).

3.2 Mozilla Public License 1.1

1. Definitions.

1.0.1. ”Commercial Use” means distribution or otherwise making the Covered Code available to a third party.

1.1. ”Contributor” means each entity that creates or contributes to the creation of Modifications.

1.2. ”Contributor Version” means the combination of the Original Code, prior Modifications used by a Contributor, and the Modifications made by that particular Contributor.

1.3. ”Covered Code” means the Original Code or Modifications or the combination of the Original Code and Modifications, in each case including portions thereof.

1.4. ”Electronic Distribution Mechanism” means a mechanism generally accepted in the software development community for the electronic transfer of data.

1.5. ”Executable” means Covered Code in any form other than Source Code.

1.6. ”Initial Developer” means the individual or entity identified as the Initial Developer in the Source Code notice required by Exhibit A.

1.7. ”Larger Work” means a work which combines Covered Code or portions thereof with code not governed by the terms of this License.

1.8. ”License” means this document.

1.8.1. ”Licensable” means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. ”Modifications” means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is:

A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.

B. Any new file that contains any part of the Original Code or previous Modifications.

1.10. ”Original Code” means Source Code of computer software code which is described in the Source Code notice required by Exhibit A as Original Code, and which, at the time of its release under this License is not already Covered Code governed by this License.

1.10.1. ”Patent Claims” means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.11. ”Source Code” means the preferred form of the Covered Code for making modifications to it, including all modules it contains, plus any associated interface definition files, scripts used to control compilation and installation of an Executable, or source code differential comparisons against either the Original Code or another well known, available Covered Code of the Contribu-

tor's choice. The Source Code can be in a compressed or archival form, provided the appropriate decompression or de-archiving software is widely available for no charge.

1.12. "You" (or "Your") means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License or a future version of this License issued under Section 6.1. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50

2. Source Code License. 2.1. The Initial Developer Grant. The Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims: (a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer to use, reproduce, modify, display, perform, sublicense and distribute the Original Code (or portions thereof) with or without Modifications, and/or as part of a Larger Work; and

(b) under Patents Claims infringed by the making, using or selling of Original Code, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Code (or portions thereof).

(c) the licenses granted in this Section 2.1(a) and (b) are effective on the date Initial Developer first distributes Original Code under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: 1) for code that You delete from the Original Code; 2) separate from the Original Code; or 3) for infringements caused by: i) the modification of the Original Code or ii) the combination of the Original Code with other software or devices.

2.2. Contributor Grant. Subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license (a) under intellectual property rights (other than patent or trademark) Licensable by Contributor, to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof) either on an unmodified basis, with other Modifications, as Covered Code and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: 1) Modifications made by that Contributor (or portions thereof); and 2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) the licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first makes Commercial Use of the Covered Code.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: 1) for any code that Contributor has deleted from the Contributor Version; 2) separate from the Contributor Version; 3) for infringements caused by: i) third party modifications of Contributor Version or ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or 4) under Patent Claims infringed by Covered Code in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Application of License. The Modifications which You create or to which You contribute are governed by the terms of this License, including without limitation Section 2.2. The Source

Code version of Covered Code may be distributed only under the terms of this License or a future version of this License released under Section 6.1, and You must include a copy of this License with every copy of the Source Code You distribute. You may not offer or impose any terms on any Source Code version that alters or restricts the applicable version of this License or the recipients' rights hereunder. However, You may include an additional document offering the additional rights described in Section 3.5.

3.2. Availability of Source Code. Any Modification which You create or to which You contribute must be made available in Source Code form under the terms of this License either on the same media as an Executable version or via an accepted Electronic Distribution Mechanism to anyone to whom you made an Executable version available; and if made available via Electronic Distribution Mechanism, must remain available for at least twelve (12) months after the date it initially became available, or at least six (6) months after a subsequent version of that particular Modification has been made available to such recipients. You are responsible for ensuring that the Source Code version remains available even if the Electronic Distribution Mechanism is maintained by a third party.

3.3. Description of Modifications. You must cause all Covered Code to which You contribute to contain a file documenting the changes You made to create that Covered Code and the date of any change. You must include a prominent statement that the Modification is derived, directly or indirectly, from Original Code provided by the Initial Developer and including the name of the Initial Developer in (a) the Source Code, and (b) in any notice in an Executable version or related documentation in which You describe the origin or ownership of the Covered Code.

3.4. Intellectual Property Matters

(a) Third Party Claims. If Contributor has knowledge that a license under a third party's intellectual property rights is required to exercise the rights granted by such Contributor under Sections 2.1 or 2.2, Contributor must include a text file with the Source Code distribution titled "LEGAL" which describes the claim and the party making the claim in sufficient detail that a recipient will know whom to contact. If Contributor obtains such knowledge after the Modification is made available as described in Section 3.2, Contributor shall promptly modify the LEGAL file in all copies Contributor makes available thereafter and shall take other steps (such as notifying appropriate mailing lists or newsgroups) reasonably calculated to inform those who received the Covered Code that new knowledge has been obtained.

(b) Contributor APIs. If Contributor's Modifications include an application programming interface and Contributor has knowledge of patent licenses which are reasonably necessary to implement that API, Contributor must also include this information in the LEGAL file.

(c) Representations. Contributor represents that, except as disclosed pursuant to Section 3.4(a) above, Contributor believes that Contributor's Modifications are Contributor's original creation(s) and/or Contributor has sufficient rights to grant the rights conveyed by this License.

3.5. Required Notices. You must duplicate the notice in Exhibit A in each file of the Source Code. If it is not possible to put such notice in a particular Source Code file due to its structure, then You must include such notice in a location (such as a relevant directory) where a user would be likely to look for such a notice. If You created one or more Modification(s) You may add your name as a Contributor to the notice described in Exhibit A. You must also duplicate this License in any documentation for the Source Code where You describe recipients' rights or ownership rights relating to Covered Code. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Code. However, You may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You

must make it absolutely clear than any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.6. Distribution of Executable Versions. You may distribute Covered Code in Executable form only if the requirements of Section 3.1-3.5 have been met for that Covered Code, and if You include a notice stating that the Source Code version of the Covered Code is available under the terms of this License, including a description of how and where You have fulfilled the obligations of Section 3.2. The notice must be conspicuously included in any notice in an Executable version, related documentation or collateral in which You describe recipients' rights relating to the Covered Code. You may distribute the Executable version of Covered Code or ownership rights under a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable version does not attempt to limit or alter the recipient's rights in the Source Code version from the rights set forth in this License. If You distribute the Executable version under a different license You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or any Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.7. Larger Works. You may create a Larger Work by combining Covered Code with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Code.

4. Inability to Comply Due to Statute or Regulation. If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Code due to statute, judicial order, or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be included in the LEGAL file described in Section 3.4 and must be included with all distributions of the Source Code. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it. 5. Application of this License. This License applies to code to which the Initial Developer has attached the notice in Exhibit A and to related Covered Code. 6. Versions of the License. 6.1. New Versions. Netscape Communications Corporation ("Netscape") may publish revised and/or new versions of the License from time to time. Each version will be given a distinguishing version number.

6.2. Effect of New Versions. Once Covered Code has been published under a particular version of the License, You may always continue to use it under the terms of that version. You may also choose to use such Covered Code under the terms of any subsequent version of the License published by Netscape. No one other than Netscape has the right to modify the terms applicable to Covered Code created under this License.

6.3. Derivative Works. If You create or use a modified version of this License (which you may only do in order to apply it to code which is not already Covered Code governed by this License), You must (a) rename Your license so that the phrases "Mozilla", "MOZILLAPL", "MOZPL", "Netscape", "MPL", "NPL" or any confusingly similar phrase do not appear in your license (except to note that your license differs from this License) and (b) otherwise make it clear that Your version of the license contains terms which differ from the Mozilla Public License and Netscape

Public License. (Filling in the name of the Initial Developer, Original Code or Contributor in the notice described in Exhibit A shall not of themselves be deemed to be modifications of this License.)

7. DISCLAIMER OF WARRANTY. COVERED CODE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED CODE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED CODE IS WITH YOU. SHOULD ANY COVERED CODE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER. 8. TERMINATION. 8.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. All sublicenses to the Covered Code which are properly granted shall survive any termination of this License. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

8.2. If You initiate litigation by asserting a patent infringement claim (excluding declatory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You file such action is referred to as "Participant") alleging that:

(a) such Participant's Contributor Version directly or indirectly infringes any patent, then any and all rights granted by such Participant to You under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively, unless if within 60 days after receipt of notice You either: (i) agree in writing to pay Participant a mutually agreeable reasonable royalty for Your past and future use of Modifications made by such Participant, or (ii) withdraw Your litigation claim with respect to the Contributor Version against such Participant. If within 60 days of notice, a reasonable royalty and payment arrangement are not mutually agreed upon in writing by the parties or the litigation claim is not withdrawn, the rights granted by Participant to You under Sections 2.1 and/or 2.2 automatically terminate at the expiration of the 60 day notice period specified above.

(b) any software, hardware, or device, other than such Participant's Contributor Version, directly or indirectly infringes any patent, then any rights granted to You by such Participant under Sections 2.1(b) and 2.2(b) are revoked effective as of the date You first made, used, sold, distributed, or had made, Modifications made by that Participant.

8.3. If You assert a patent infringement claim against Participant alleging that such Participant's Contributor Version directly or indirectly infringes any patent where such claim is resolved (such as by license or settlement) prior to the initiation of patent infringement litigation, then the reasonable value of the licenses granted by such Participant under Sections 2.1 or 2.2 shall be taken into account in determining the amount or value of any payment or license.

8.4. In the event of termination under Sections 8.1 or 8.2 above, all end user license agreements (excluding distributors and resellers) which have been validly granted by You or any distributor hereunder prior to termination shall survive termination.

9. LIMITATION OF LIABILITY. UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF

COVERED CODE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU. 10. U.S. GOVERNMENT END USERS. The Covered Code is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Code with only those rights set forth herein. 11. MISCELLANEOUS. This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by California law provisions (except to the extent applicable law, if any, provides otherwise), excluding its conflict-of-law provisions. With respect to disputes in which at least one party is a citizen of, or an entity chartered or registered to do business in the United States of America, any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California, with venue lying in Santa Clara County, California, with the losing party responsible for costs, including without limitation, court costs and reasonable attorneys' fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. 12. RESPONSIBILITY FOR CLAIMS. As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability. 13. MULTIPLE-LICENSED CODE. Initial Developer may designate portions of the Covered Code as Multiple-Licensed. Multiple-Licensed means that the Initial Developer permits you to utilize portions of the Covered Code under Your choice of the NPL or the alternative licenses, if any, specified by the Initial Developer in the file described in Exhibit A.

EXHIBIT A -Mozilla Public License.

"The contents of this file are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/MPL/

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specificlanguage governing rights and limitations under the License.

The Original Code is DEOS code.

The Initial Developer of the Original Code is Università degli Studi di Lecce. Portions created by the Initial Developer are Copyright (C) 2004 the Initial Developer. All Rights Reserved.

Contributor(s): _.

Alternatively, the contents of this file may be used under the terms of the _ license (the [_] License), in which case the provisions of [_] License are applicable instead of those above. If you wish to allow use of your version of this file only under the terms of the [_] License and not to allow others to use your version of this file under the MPL, indicate your decision by deleting the provisions above and replace them with the notice and other provisions required by the [_] License. If you do not delete the provisions above, a recipient may use your version of this file under either the MPL or the [_] License."

[NOTE: The text of this Exhibit A may differ slightly from the text of the notices in the Source Code files of the Original Code. You should use the text of this Exhibit A rather than the text found in the Original Code Source Code for Your Modifications.]

3.3 The Academic Free License v. 2.1

This Academic Free License (the "License") applies to any original work of authorship (the "Original Work") whose owner (the "Licensor") has placed the following notice immediately following the copyright notice for the Original Work:

Licensed under the Academic Free License version 2.1

1) Grant of Copyright License. Licensor hereby grants You a world-wide, royalty-free, non-exclusive, perpetual, sublicenseable license to do the following:

a) to reproduce the Original Work in copies;

b) to prepare derivative works ("Derivative Works") based upon the Original Work;

c) to distribute copies of the Original Work and Derivative Works to the public;

d) to perform the Original Work publicly; and

e) to display the Original Work publicly.

2) Grant of Patent License. Licensor hereby grants You a world-wide, royalty-free, non-exclusive, perpetual, sublicenseable license, under patent claims owned or controlled by the Licensor that are embodied in the Original Work as furnished by the Licensor, to make, use, sell and offer for sale the Original Work and Derivative Works.

3) Grant of Source Code License. The term "Source Code" means the preferred form of the Original Work for making modifications to it and all available documentation describing how to modify the Original Work. Licensor hereby agrees to provide a machine-readable copy of the Source Code of the Original Work along with each copy of the Original Work that Licensor distributes. Licensor reserves the right to satisfy this obligation by placing a machine-readable copy of the Source Code in an information repository reasonably calculated to permit inexpensive and convenient access by You for as long as Licensor continues to distribute the Original Work, and by publishing the address of that information repository in a notice immediately following the copyright notice that applies to the Original Work.

4) Exclusions From License Grant. Neither the names of Licensor, nor the names of any contributors to the Original Work, nor any of their trademarks or service marks, may be used to endorse or promote products derived from this Original Work without express prior written permission of the Licensor. Nothing in this License shall be deemed to grant any rights to trademarks, copyrights, patents, trade secrets or any other intellectual property of Licensor except as expressly stated herein. No patent license is granted to make, use, sell or offer to sell embodiments of any patent claims other than the licensed claims defined in Section 2. No right is granted to the trademarks of Licensor even if such marks are included in the Original Work. Nothing in this License

shall be interpreted to prohibit Licensor from licensing under different terms from this License any Original Work that Licensor otherwise would have a right to license.

5) This section intentionally omitted.

6) Attribution Rights. You must retain, in the Source Code of any Derivative Works that You create, all copyright, patent or trademark notices from the Source Code of the Original Work, as well as any notices of licensing and any descriptive text identified therein as an "Attribution Notice." You must cause the Source Code for any Derivative Works that You create to carry a prominent Attribution Notice reasonably calculated to inform recipients that You have modified the Original Work.

7) Warranty of Provenance and Disclaimer of Warranty. Licensor warrants that the copyright in and to the Original Work and the patent rights granted herein by Licensor are owned by the Licensor or are sublicensed to You under the terms of this License with the permission of the contributor(s) of those copyrights and patent rights. Except as expressly stated in the immediately proceeding sentence, the Original Work is provided under this License on an "AS IS" BASIS and WITHOUT WARRANTY, either express or implied, including, without limitation, the warranties of NON-INFRINGEMENT, MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY OF THE ORIGINAL WORK IS WITH YOU. This DISCLAIMER OF WARRANTY constitutes an essential part of this License. No license to Original Work is granted hereunder except under this disclaimer.

8) Limitation of Liability. Under no circumstances and under no legal theory, whether in tort (including negligence), contract, or otherwise, shall the Licensor be liable to any person for any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or the use of the Original Work including, without limitation, damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses. This limitation of liability shall not apply to liability for death or personal injury resulting from Licensor's negligence to the extent applicable law prohibits such limitation. Some jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, so this exclusion and limitation may not apply to You.

9) Acceptance and Termination. If You distribute copies of the Original Work or a Derivative Work, You must make a reasonable effort under the circumstances to obtain the express assent of recipients to the terms of this License. Nothing else but this License (or another written agreement between Licensor and You) grants You permission to create Derivative Works based upon the Original Work or to exercise any of the rights granted in Section 1 herein, and any attempt to do so except under the terms of this License (or another written agreement between Licensor and You) is expressly prohibited by U.S. copyright law, the equivalent laws of other countries, and by international treaty. Therefore, by exercising any of the rights granted to You in Section 1 herein, You indicate Your acceptance of this License and all of its terms and conditions.

10) Termination for Patent Action. This License shall terminate automatically and You may no longer exercise any of the rights granted to You by this License as of the date You commence an action, including a cross-claim or counterclaim, against Licensor or any licensee alleging that the Original Work infringes a patent. This termination provision shall not apply for an action alleging patent infringement by combinations of the Original Work with other software or hardware.

11) Jurisdiction, Venue and Governing Law. Any action or suit relating to this License may be brought only in the courts of a jurisdiction wherein the Licensor resides or in which Licensor conducts its primary business, and under the laws of that jurisdiction excluding its conflict-of-law provisions. The application of the United Nations Convention on Contracts for the International

Sale of Goods is expressly excluded. Any use of the Original Work outside the scope of this License or after its termination shall be subject to the requirements and penalties of the U.S. Copyright Act, 17 U.S.C. Â§ 101 et seq., the equivalent laws of other countries, and international treaty. This section shall survive the termination of this License.

12) Attorneys Fees. In any action to enforce the terms of this License or seeking damages relating thereto, the prevailing party shall be entitled to recover its costs and expenses, including, without limitation, reasonable attorneys' fees and costs incurred in connection with such action, including any appeal of such action. This section shall survive the termination of this License.

13) Miscellaneous. This License represents the complete agreement concerning the subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable.

14) Definition of "You" in This License. "You" throughout this License, whether in upper or lower case, means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, "You" includes any entity that controls, is controlled by, or is under common control with you. For purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50

15) Right to Use. You may use the Original Work in all ways not otherwise restricted or conditioned by this License or by law, and Licensor promises not to interfere with or be responsible for such uses by You.

Bibliography

[1] Narayanan, A. et al., "Research in Object-Oriented Manufacturing Simulations: An Assessment of the State of the Art." *IIE Transactions*, 1998