Developments in Integration of Simulation Tools

John G Pearce
ISIM International Simulation Limited
161 Claremont Road
Salford, M6 8PA, UK
John.Pearce@isimsimulation.com

Ryllan J Kraft
ISIM International Simulation Limited
1 St Crispins Close
London, NW3 2QF, UK
Ryllan.Kraft@isimsimulation.com

Key Words: Multi-discipline**,** Multi-tool, Large-scale, ESL, VTB

Abstract

The paper presents recent developments in an ongoing project to integrate the ESL simulation tool with the Virtual Test Bed (VTB). The VTB offers a framework for the development of large-scale multi-discipline simulations. It provides an environment in which to build a simulation and is capable of supporting different solution methods (or "Solvers"). The objective of the present project is to make the particular features of ESL available to users of the VTB through the provision of an ESL Solver. This is achieved by making ESL simulation components available in VTB and seamlessly linking to the external ESL software for compilation and execution, with data being fed back to the VTB for display. A previous paper described the basic approach and initial progress – the present paper reports further progress and concentrates on *modularization* aspects. The VTB provides a means of partitioning a large schematic diagram into interconnected subsystems. This feature works quite naturally with ESL schematics. A standard mechanism is also provided for creating multi-input, multi-output modules from schematic diagrams comprising interconnected simulation components. This facility can be used to map a VTB schematic comprising interconnected ESL components into a VTB module, implemented as an ESL submodel, which can be used in other schematics. A third strand has been the implementation of a method to import external ESL textual submodels into the VTB. All these developments represent a significant advancement of the integration of the two simulation tools and provide valuable pointers to a generalized approach for other solvers. The developments are illustrated with appropriate examples.

1. INTRODUCTION

The simulation of very large complex multi-discipline systems requires the use of a wide range of different tools. Specialized circuit modelling tools may be appropriate for those parts of the system comprising electrical components, whereas other tools are more suited to dynamics and mechanical areas of the system. Even within a given discipline, different types of model are used – natural coupled schematic representations; signal flow diagrams; state-space and differential equation representations. The objective of the current research is to demonstrate that different simulation tools can be integrated in a manner which provides the user with a unified environment from which the tools can be accessed in a natural and convenient manner. In order to achieve this objective, an initiative has been undertaken to integrate two specific simulation tools – the USC Virtual Test Bed (VTB) and ISIM's ESL simulation tool.

The VTB [Dougal 2005] is a software environment for developing simulations of large scale multidisciplinary dynamic systems. It allows alternative designs to be analysed and tested before being committed to manufacture. The main application that is driving the development of the VTB is a need to model advanced power systems for navy ships. In such systems there are many different energy generation and storage devices including nuclear, fuel cells and gas turbines. The distribution networks are also of unconventional design having DC power buses and high numbers of interconnections that can be rapidly reconfigured. The VTB provides a number of in-built solvers, offering alternative solution methods, however, the particular feature that commends it to this study is the provision an interface for user created solvers. Thus it is relatively easy to introduce alternative methods of both system specification and simulation solution.

ESL [Crosbie et al 1981, Hay et al 1994, Pearce and Crosbie 2000] is an advanced high-level simulation language for modelling large-scale systems from a variety of disciplines. ESL comprises two components: the language itself and a graphical user interface - the Integrated Simulation Environment (ISE). ESL is a continuous system simulation language and is used for modelling non-linear dynamic systems which are usually described by ordinary and partial differential equations. It has advanced features for accurately processing discontinuities and the capability of implementing multi-rate simulations. ISE provides the environment from which all stages of the simulation process can be managed. The software was developed mainly through a series of contracts with ESTEC - the European Space Research and Technology Centre - part of ESA with additional support from various industrial simulation consultancy activities.

The first stage of the integration project saw the establishment of basic essential functionality – the provision of ESL entities in VTB and a transparent mechanism of interfacing with the external ESL software. The present paper is concerned with aspects of *modularization* and demonstrates how VTB's inherent modular structures may be mapped onto ESL's submodel structures.

2. EARLIER WORK

Earlier work [Pearce, 2007, 2008] introduced the ability to include complete ESL models in VTB schematics using COM technology (with VTB 2003) and later .NET technology (VTB Pro. and VTB 2009) as self-contained VTB signal entities. However such ESL models could only be coupled to other signal entities and gave the user no access to their internal structure from the VTB.

As part of a current ONR funded research project, ways of achieving greater integration between simulation tools are being investigated through the particular example of the VTB – ESL relationship. These developments are based on VTB 2011. During the first phase of this project the following objectives have been achieved [Pearce and Kraft 2011]:

- Provision of a set of entities in the VTB corresponding to the standard ESL simulation elements. These are displayed in the VTB Schematic Editor when the ESL Solver is selected on the Component Library filter.
- Ability to create a schematic diagram by placing and connecting ESL entities (in this phase of the project there is no connection to non ESL entities).
- Ability to specify ESL entity parameters (corresponding to ESL simulation element attributes).
- Ability to select ESL entity input and output ports for graph plotting.
- Ability to specify ESL Solver parameters (corresponding to ESL Simulation Parameters – integration method, error tolerances number of integration steps per communication interval etc).
- Interactive running of the simulation including the ability to pause and dynamically change ESL entity and ESL Solver parameters.
- Ability to extend the standard set of ESL entities by specifying the ESL code associated with a new entity through an XML attribute.

The overall scheme of achieving these objectives is illustrated in Figure 1. Portions of a VTB schematic diagram comprising ESL components are recognised and mapped into ESL source code which is externally compiled and packaged as a .NET assembly. During the running of the simulation, the assembly appears to the VTB as a single component communicating with the rest of the schematic at step points. The whole of this process is fully automatic and takes place behind the scenes once a simulation run is initiated.

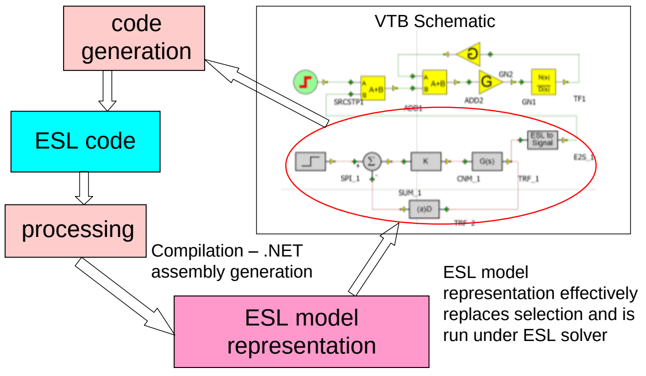

code generation
ESL code
processing
Compilation – .NET assembly generation
ESL model representation
VTB Schematic
ESL model representation effectively replaces selection and is run under ESL solver

Figure 1 VTB – ESL Integration

3. MODULARIZATION

As it would be cumbersome to represent a very large simulation as a single schematic diagram, procedures are provided to partition a large system diagram into smaller, more manageable units.

In VTB the user has the option to break a large schematic diagram into interconnected subsystems. Special *connector* entities are provided for this purpose to specify the connection points between subsystems. Also in VTB, sections of a schematic diagram can be converted into self-contained *modules* which are then represented as single icons. An advantage of modules is that multiple-instances of a module, each with its own set of parameters, can appear in a system or subsystem diagram.

ESL does not have the equivalence of subsystems, but it does have the concept of *submodels* which are similar to VTB modules (but without the ability to assign parameters). A submodel may be defined graphically as a block diagram, or textually in the underlying ESL language. Thus the next stage of integration was to map the ESL submodel concept into VTB's subsystem and module facilities.

3.1. Integration – subsystems

Since schematic diagrams comprising only ESL components are essentially no different to schematics of other types of VTB components, the VTB subsystem facility can be used to break ESL schematics into smaller units without the need for software modifications. Figure 2 and Figure 3 show, as an example, an ESL schematic of a servo system split into two subsystems. Connections between the subsystems are made through the three connectors.

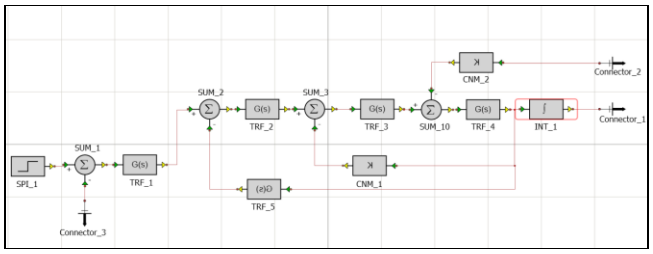


Figure 2 – Servo Subsystem 1

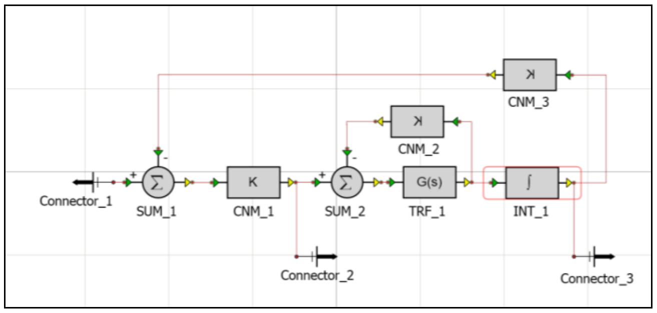


Figure 3 – Servo Subsystem 2

3.2. Integration – ESL graphical submodels

The VTB software includes a Module Designer which is used to create modules from schematic diagrams. Figure 4 shows the ESL schematic of the complete servo system used in the previous example, as it appears in VTB, with the *controller*, *motor*, and *gearbox* sections identified for conversion into modules.

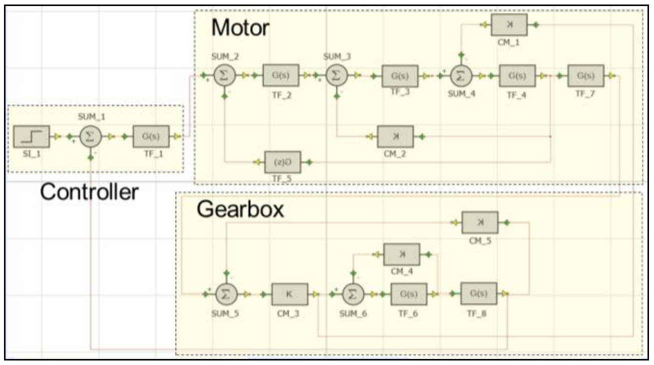


Figure 4 - Complete Servo System

The procedure is to import a section of diagram into Module Designer, where both ports and parameters may be exposed. This is shown in Figure 5 for the *motor* section of the diagram. The same procedure is followed for the other sections. Any nodes on the diagram may be exposed and named as module ports. Parameters of any component may be also renamed and exposed as parameters of the module.

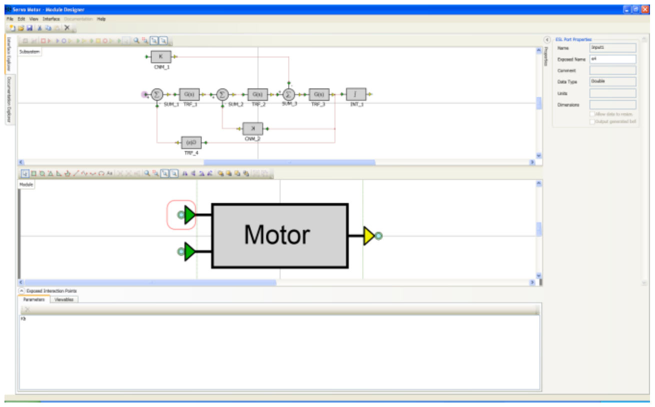


Figure 5 - Motor module being developed

Figure 6 shows the servo system reconstructed used the modules created using Module Designer. Two servo systems are created (with different parameters) to demonstrate the use of multiple-instances of a single module.

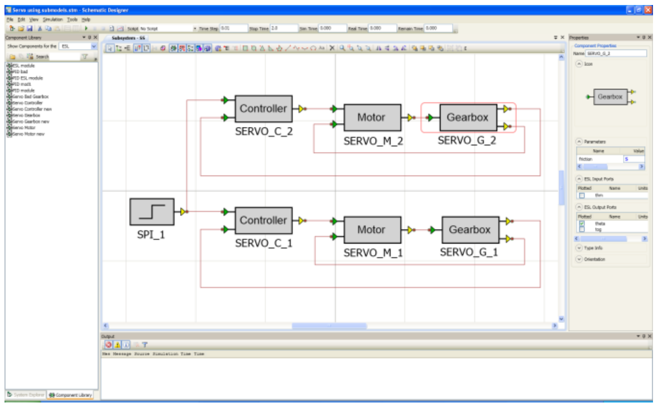


Figure 6 - Dual Servo System rebuilt with modules

ESL Solver (the VTB add-in that processes ESL components) now has the additional task of identifying module definitions (which are converted into ESL submodels) and module instances (which are converted into submodel calls. Figure 7 presents fragments of ESL code generated from the servo example. The definition of the *Servo_Controller* submodel and the two instances of its call are highlighted.

```
-- Created by ESL-VTB on 2011-11-19 11:43

EMBEDDED

SUBMODEL Servo_Controller(REAL: e4 := REAL: theta_d; REAL:
theta);
  REAL: SS_SUM_1_x;
  REAL: SS_SUM_1_y;
  REAL: SS_TRF_1_y;
  REAL: SS_SUM_1_z;
DYNAMIC
  e4 := SS_TRF_1_y;
  SS_SUM_1_x := theta_d;
  SS_SUM_1_y := theta;
  SS_TRF_1_y := TRANSFER(27.0(12.8+s)/s(1+0.001*s)) *
SS_SUM_1_z;
  SS_SUM_1_z := SS_SUM_1_x + (-SS_SUM_1_y); -- x + y
END Servo_Controller;
.................
.................

INCLUDE %stepp%;

PACKAGE Esl_Io;
.................
.................
END Esl_Io;
SEGMENT EslGenerate;
.................
.................
DYNAMIC
  SS_SPI_1_LOG := STEPP(SS_SPI_1_TD);
  SS_SPI_1_y := IF SS_SPI_1_LOG THEN SS_SPI_1_K ELSE 0.0;
  SS_SERVO_C_1_e4 := Servo_Controller(SS_SPI_1_y,
SS_SERVO_G_1_theta);
  SS_SERVO_G_1_theta, SS_SERVO_G_1_tog :=
                      Servo_Gearbox(SS_SERVO_M_1_thm,
SS_SERVO_G_1_friction);
  SS_SERVO_C_2_e4 := Servo_Controller(SS_SPI_1_y,
SS_SERVO_G_2_theta);
  SS_SERVO_G_2_theta, SS_SERVO_G_2_tog :=
                      Servo_Gearbox(SS_SERVO_M_2_thm,
SS_SERVO_G_2_friction);
  SS_SERVO_M_1_thm :=
  Servo_Motor(SS_SERVO_C_1_e4, SS_SERVO_G_1_tog,
SS_SERVO_M_1_Kb);
  SS_SERVO_M_2_thm :=
  Servo_Motor(SS_SERVO_C_2_e4, SS_SERVO_G_2_tog,
SS_SERVO_M_2_Kb);
.................
.................
END EslGenerate;
```

Figure 7 - ESL code using submodels

3.3. Integration – textual submodels

A stated objective of the ESL-VTB integration was to provide a means of including ESL language code as part of a simulation. This could be achieved by providing a means of importing ESL textual submodels into a VTB schematic diagram. Although this was already possible through the general ability to extend the standard set of ESL entities (a user could specify a submodel call using the *EslEntityXml* parameter in Entity Designer – see [Pearce and Kraft 2011]), an automated method has been implemented in the form of an *ESL Submodel to VTB Entity* tool (shown in Figure 9). This gives the VTB user direct access to both existing ESL submodel libraries and newly created submodels.

The user invokes the tool, and can press the *Load ESL Submodels* button to select one or more files of ESL source code containing submodels. The tool identifies them, and their interface - the input and output arguments of the submodel. The user can click the *Create VTB Entities* button to initiate the creation of VTB entity components for the submodels, which are installed into a deployment folder for use by the VTB Entity Designer and Schematic Designer.

The tool creates a functional basic block icon with the appropriate ports corresponding to the inputs and outputs. However, it is expected that users will want to make use of the VTB Entity Designer to tailor the entity's icon to improve its appearance and to suit their requirements. Prior to the creation, the user can specify a number of attributes for the submodels, such as the name and prefix for the desired VTB entity and the names in VTB for the input and output ports. If an input to the submodel has been declared as *Constant* in the ESL source code it will be expressed as a parameter of the VTB entity. Otherwise the user may select whether the input should be a parameter or be an input port for the VTB entity. In addition, default values may be specified for parameters.

As an example, Figure 8 shows the ESL source code for a simple linear dc motor submodel. Note that *Constant* input arguments *La*, *Ja* and *Ba* will be pre-selected in the tool as parameters, whereas the user has the choice of allowing the remaining input arguments to map to entity input ports or selecting them as parameters.

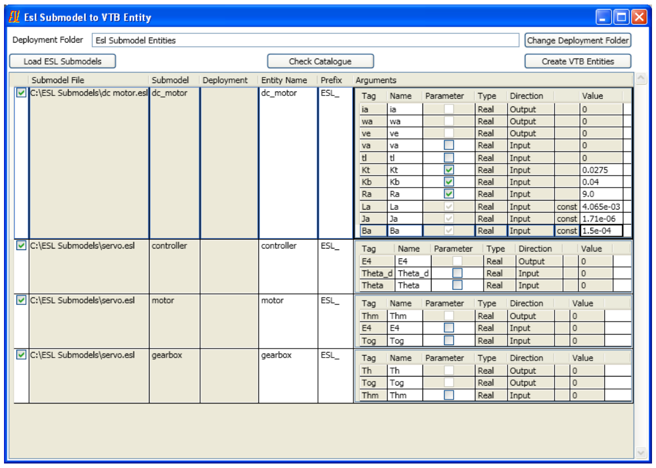
```
Submodel dc_motor(Real: ia, wa, ve := Real: va, tl, Kt, Kb, Ra;
Constant Real: La, Ja, Ba);
        Real: vb, tm, ta;

        Initial

        Dynamic
                ve := va - vb;
                ia := Transfer(1/(La*s + Ra))*ve;
                tm := Kt*ia;
                ta := tm - tl;
                wa := Transfer(1/(Ja*s + Ba))*ta;
                vb := Kb*wa;
        End dc_motor;
```

Figure 8 - ESL code for dc motor

Figure 9 shows the tool with the dc motor submodel file loaded (along with a second file containing the three servo system submodels). The entity name, prefix and the names of the input and output ports have been changed; input arguments *Kt*, *Kb* and *Ra* have been selected as entity parameters and all parameters have been assigned default values. All submodels are selected (as shown by the left-had side checkboxes) for VTB entity creation.



Esl Submodel to VTB Entity

Deployment Folder: Esl Submodel Entities

	Submodel File	Submodel	Deployment	Entity Name	Prefix
✓	C:\ESL Submodels\dc motor.esl	dc_motor		dc_motor	ESL_
✓	C:\ESL Submodels\servo.esl	controller		controller	ESL_
✓	C:\ESL Submodels\servo.esl	motor		motor	ESL_
✓	C:\ESL Submodels\servo.esl	gearbox		gearbox	ESL_

Arguments (dc_motor):

Tag	Name	Parameter	Type	Direction		Value
ia	ia		Real	Output		0
wa	wa		Real	Output		0
ve	ve		Real	Output		0
va	va	☐	Real	Input		0
tl	tl	☐	Real	Input		0
Kt	Kt	✓	Real	Input		0.0275
Kb	Kb	✓	Real	Input		0.04
Ra	Ra	✓	Real	Input		9.0
La	La	✓	Real	Input	const	4.065e-03
Ja	Ja	✓	Real	Input	const	1.71e-06
Ba	Ba	✓	Real	Input	const	1.5e-04

Arguments (controller):

Tag	Name	Parameter	Type	Direction		Value
E4	E4		Real	Output		0
Theta_d	Theta_d	☐	Real	Input		0
Theta	Theta	☐	Real	Input		0

Arguments (motor):

Tag	Name	Parameter	Type	Direction		Value
Thm	Thm		Real	Output		0
E4	E4	☐	Real	Input		0
Tog	Tog	☐	Real	Input		0

Arguments (gearbox):

Tag	Name	Parameter	Type	Direction		Value
Th	Th		Real	Output		0
Tog	Tog		Real	Output		0
Thm	Thm	☐	Real	Input		0

Figure 9 - ESL Submodel to VTB Entity tool

Figure 10 shows the created entity as it would appear in Entity Designer, at which stage the icon appearance could be tailored if desired.

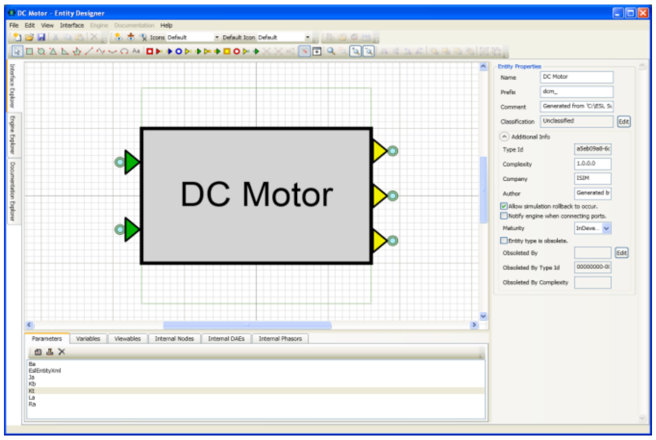


Figure 10 - dc motor entity in Entity Designer

Figure 11 shows the newly created entity in use in Schematic Designer. The entity is selected and its parameters are displayed in the right-hand panel.

One significant advantage of mapping ESL submodels (whether graphical or textual) into VTB modules or entities is the exposure of parameters, a feature currently not available in the ESL simulation environment.

The tool creates the VTB entity by forming a *VTE* file – using the VTB entity name and with extension .vte which the VTB Entity and Schematic Designers can read. It forms the file by inserting specific information – obtained from the ESL submodel code and supplemented by user input – into a template. The template is provided in the form of an external file *vte-template.xml* located in the same directory as the tool. The created entity VTE file is deployed by putting it in a subdirectory under the user's Personal (Documents and Settings) named:

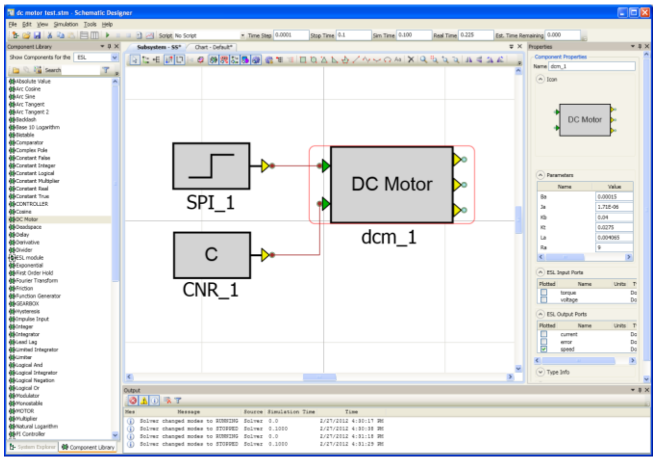
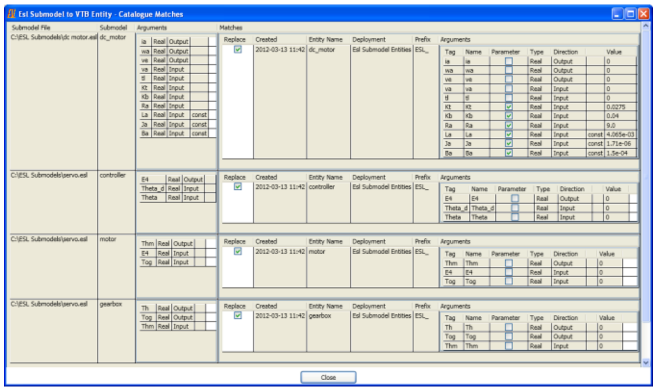


Figure 11 - Entity used in Schematic Designer

"VTB\CommonFolders\Engines\Esl Submodel Entities" – where the VTB Designers look for entity components. The name of the deployment folder may be changed by the user, for example, to keep together submodel entities relating to a particular project.

The tool maintains a *catalogue* of submodels (Figure 12) for which it has created VTB entities. When the user loads one or more files, the tool checks to see if any file and submodel pair match any in its catalogue. If so it presents a *Catalogue Matches* dialog which allows the user to select to replace the VTB entity with the current one when the *Create VTB Entities* button is pressed. Any settings that may have been changed when the previous entity was created, such as entity name, prefix and argument settings will be picked up if they have a correspondence in the new submodel. The user may press a *Check Catalogue* button at any time to invoke this dialog on all loaded submodels to remove or change the current *replacements*. If no replacement is specified when the *Create VTB Entities* button is pressed, a new VTB entity is created and deployed.



Esl Submodel to VTB Entity - Catalogue Matches

Submodel File	Submodel	Replace	Created	Entity Name	Deployment	Prefix
C:\ESL Submodels\dc motor.esl	dc_motor	✓	2012-03-13 11:42	dc_motor	Esl Submodel Entities	ESL_
C:\ESL Submodels\servo.esl	controller	✓	2012-03-13 11:42	controller	Esl Submodel Entities	ESL_
C:\ESL Submodels\servo.esl	motor	✓	2012-03-13 11:42	motor	Esl Submodel Entities	ESL_
C:\ESL Submodels\servo.esl	gearbox	✓	2012-03-13 11:42	gearbox	Esl Submodel Entities	ESL_

Arguments (dc_motor):

ia	Real	Output	
wa	Real	Output	
ve	Real	Output	
va	Real	Input	
tl	Real	Input	
Kt	Real	Input	
Kb	Real	Input	
Ra	Real	Input	
La	Real	Input	const
Ja	Real	Input	const
Ba	Real	Input	const

Matches – Arguments (dc_motor):

Tag	Name	Parameter	Type	Direction		Value
ia	ia	☐	Real	Output		0
wa	wa	☐	Real	Output		0
ve	ve	☐	Real	Output		0
va	va	☐	Real	Input		0
tl	tl	☐	Real	Input		0
Kt	Kt	✓	Real	Input		0.0275
Kb	Kb	✓	Real	Input		0.04
Ra	Ra	✓	Real	Input		9.0
La	La	✓	Real	Input	const	4.065e-03
Ja	Ja	✓	Real	Input	const	1.71e-06
Ba	Ba	✓	Real	Input	const	1.5e-04

Arguments (controller):

E4	Real	Output
Theta_d	Real	Input
Theta	Real	Input

Matches – Arguments (controller):

Tag	Name	Parameter	Type	Direction		Value
E4	E4	☐	Real	Output		0
Theta_d	Theta_d	☐	Real	Input		0
Theta	Theta	☐	Real	Input		0

Arguments (motor):

Thm	Real	Output
E4	Real	Input
Tog	Real	Input

Matches – Arguments (motor):

Tag	Name	Parameter	Type	Direction		Value
Thm	Thm	☐	Real	Output		0
E4	E4	☐	Real	Input		0
Tog	Tog	☐	Real	Input		0

Arguments (gearbox):

Th	Real	Output
Tog	Real	Output
Thm	Real	Input

Matches – Arguments (gearbox):

Tag	Name	Parameter	Type	Direction		Value
Th	Th	☐	Real	Output		0
Tog	Tog	☐	Real	Output		0
Thm	Thm	☐	Real	Input		0

Figure 12 - Entity Catalogue

4. THE NEXT STAGE

The major outstanding task which is currently being addressed in this project is to allow the construction and execution of a schematic comprising both *coupled* ESL components and non-ESL components (i.e. those requiring a different solver). This will require special *crossover* components having both ESL and non-ESL ports (signal ports, for example). Once completed, this will enable truly integrated simulations to be developed – using both native VTB solvers and external third-party simulation software.

5. CONCLUSION

Large-scale simulations require the use of widely differing modelling tools. As part of an on-going research project further aspects of the integration of different simulation modelling tools have been examined with particular emphasis on modularization. Using the example the integration of ESL and the VTB, techniques of mapping ESL's submodel structures onto VTB modules have been described. Graphically described ESL submodels are mapped into VTB modules whereas external ESL textual submodels are converted to VTB entities. The overall objective of providing users with access to different simulation tools from a single environment has been advanced.

Acknowledgements

This research has been supported by the Office of Naval Research through Award No. N00014-10-1-0625. The authors also acknowledge the contributions of Blake Langland and Rod Leonard of the University of South Carolina.

References

Crosbie, R.E., Hay, J.L. and Pearce, J.G. 1981. "Simulation Studies with Modern Computer Structures". Final Report, (ESTEC Contract 4155/79), ESTEC, Noordwijk, The Netherlands.

Dougal, R.A. 2005. "Design Tools for Electric Ship Systems." In Proceedings of IEEE Electric Ship Technologies Symposium, (Philadelphia PA, July 25-27). IEEE, 8-11.

Hay, J.L., Pearce, J.G., Crosbie, R.E. and Pallett, S. 1994. "ESL Simulation Tool". Final Report, (ESTEC Contract 10011/92/NL/JG Work Order No. 2), ESTEC, Noordwijk, The Netherlands.

Pearce, J.G. 2007. "Interfacing the ESL Simulation Language to the Virtual Test Bed". In Proceedings of the 2007 Western Multiconference on Computer Simulation, (San Diego, CA, Jan 14-17). SCS, San Diego, CA, 166-171.

Pearce, J.G. 2008. "Simulation advances using the ESL Simulation Language and the Virtual Test Bed". In Proceedings of the 2008 Grand Challenges in Modeling & Simulation Conference (GCMS), (Edinburgh, Scotland UK, June 16-18). SCS, San Diego, CA, 285-290.

Pearce, J.G. and Crosbie, R.E. 2000. "ESL-ISE - A Simulation Tool Developed for the Space Industry". In Proceedings of the 2000 International Conference on Simulation and Multimedia in Engineering Education, (San Diego, CA, Jan 23-27). SCS, San Diego, CA, 115-120.

Pearce, J.G. and Kraft, R.J. 2011. "Multi-discipline, Multi-tool Simulation Developments" In Proceedings of the 2011 Grand Challenges in Modeling & Simulation Conference (GCMS), (The Hague, Netherlands, June 27-29 June). SCS, Vista, CA, 246-251.

Biography

John Pearce received his BSc in Electrical Engineering in 1970 and PhD in Computer Simulation in 1973 from the University of Salford, UK. He held a Research Fellowship for four years in the Department of Electrical Engineering at the University of Salford where he worked on the of Simulation of Atomic Collision Processes and the development of Continuous System Simulation Languages. This was followed by a period of some twenty-six years as a full-time member of academic staff in the same department. He continued to develop his interest in System Simulation and CSSLs and, together with John Hay and Roy Crosbie, developed the ISIS and ISIM simulation languages. From 1986 he has been associated with ISIM International Simulation Limited where he contributed to a series of research contracts with the European Space Agency leading to was the creation of the ESL simulation language. From 2007 he has contributed to a series of simulation projects funded by the US Office of Naval Research through California State University, Chico and the University of South Carolina. This has included providing support for multi-rate simulation, and the integration of ESL with the USC's Virtual Test Bed (VTB) software. He also continues to work part-time as a lecturer at the University of Salford.

Ryllan Kraft graduated in Natural Sciences, specialising in Physics, from Cambridge University, UK and went on to do research at the Institute of Astronomy. Following that he went into commercial computing consultancy and worked for some years in California, USA, and in Sweden. Returning to the UK, he led a team developing real-time industrial knowledge based systems for many years, and in the course of that worked with John Pearce and ISIM to integrate simulations with knowledge based systems. More recently he has worked in the area of computer based gaming. The areas of computing he has worked on include image analysis, graphics applications, real-time telecommunications, knowledge engineering & knowledge based systems, databases and computer games.