Multi-discipline, Multi-tool Simulation Developments

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 addresses issues relating to the integration of simulation tools with the objective of providing greater flexibility and efficiency of model development. A specific example of a current research activity aimed at incorporating the functionality of the ESL simulation tool within the Virtual Test Bed (VTB) is described. A brief introduction to ESL and VTB is presented in which the philosophies and strengths of the two packages are described, together with a summary of earlier work resulting in the ability to incorporate externally generated ESL models within VTB simulations. This leads to the specification of an initial set of goals aimed at achieving much greater integration of the two simulation packages – the ultimate objective being to replicate the functionality of ESL within the VTB. A description of underlying principles of the method being developed to achieve these goals is presented. Essentially this involves the inclusion of the set of basic ESL simulation elements with the VTB allowing the construction of ESL style block diagrams. Infrastructure is provided to map the ESL block diagrams into a form that can be executed by the VTB as any other VTB simulation. The current stage of the work is illustrated with a simple example and a set of further goals aimed at extending the ESL functionality available from the VTB is presented. The potential improvements in efficiency and effectiveness of simulation development through such integrated environments are discussed.

1. INTRODUCTION

Large-scale simulations often require the integration of numerous models developed using diverse tools and programming languages. For example tools for implementing electronic circuit simulations are not suitable for modelling mechanical dynamics. Control engineers also have their preferred software packages. The *grand challenge* is to integrate these diverse models within a single unified environment from which the whole simulation can be managed. This poses many problems: different simulation tools will generate models in different target code; mechanisms for interacting with individual models will vary considerably; the resolution or granularity of different models will also vary.

An on-going research project which aims to address these issues is the development of the Virtual Test Bed (VTB) by the University of South Carolina, USA [Dougal 2005]. The VTB 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. Constructing complete coherent simulations of such large scale systems, involving widely differing technologies poses a serious challenge. Each discipline group will use their preferred simulation tool to model their part of the whole system. The VTB aims to satisfy this challenge by providing a common platform in which entity models developed by different teams using different tools can be merged.

Complete models developed in their native environments outside the VTB can be imported and combined with other models. However, the VTB provides a means of achieving greater integration by supporting a range of internal *solvers*. Typically a VTB simulation is constructed by interconnecting primary simulation elements or *entities*. Each type of entity is supported by its own solver. For example, The VTB provides: *DAE*, *Phasor*, *signal* and *Natural* solvers as standard, but the developer is at liberty to introduce his own solvers to support his own flavour of entities. This solver concept allows for greater integration of external simulation tools with the VTB. This paper describes current developments to integrate the ESL simulation tool with the VTB.

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 is an acronym of the *European Simulation Language* (originally European Space Agency Simulation Language) and 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 dynamic systems which are usually described by ordinary and partial differential equations. 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.

2. BACKGROUND

Previous work [Pearce, 2007, 2008] saw the ability to include complete ESL models in VTB schematics using COM technology (with VTB 2003) and later .NET technology (VTB 2009 and VTB Pro). Here the starting point is an ESL model, which may have been created as ESL source code or via ESL's Integrated Simulation Environment. The model is cast as an *embedded segment* (a particular form of model that can be readily embedded in other non-ESL programs). The embedded segment is compiled as a .NET assembly which is then processed by the VTB Entity Designer program to create an ESL Entity. The ESL Entity is then accessible from the VTB Schematic Designer and can be included in a schematic diagram with other non-ESL entities. The Entity creation process for a dc motor model is shown in Figure 1.

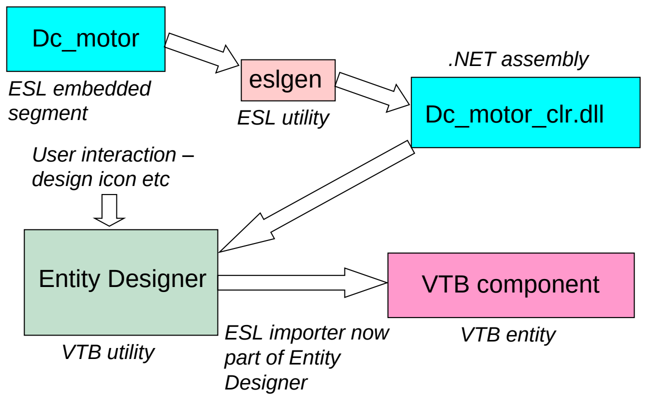


Figure 1 ESL Entity creation process

At simulation run-time, the VTB invokes the ESL Entity at each time-step to advance its part of the solution, as shown in Figure 2

At this stage complete ESL models, created outside VTB, may be combined with non-ESL entities in a whole VTB schematic to create a complete multi-solver simulation.

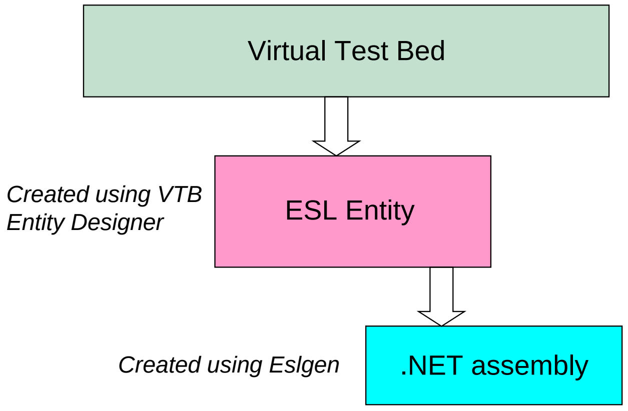


Figure 2 VTB – ESL .NET execution sequence

3. TOWARDS GREATER INTEGRATION

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.

A primary objective was to replicate in the VTB a subset of the functionality currently available in ESL. This entailed:

- Provision of a set of entities in the VTB corresponding to the standard ESL simulation elements. These would be displayed in the VTB Schematic Editor when the ESL Solver was selected on the Component Library filter.
- Ability to create a schematic diagram by placing and connecting ESL entities (for the primary objective there would be 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.

4. INTEGRATION SOLUTION

4.1. Basics

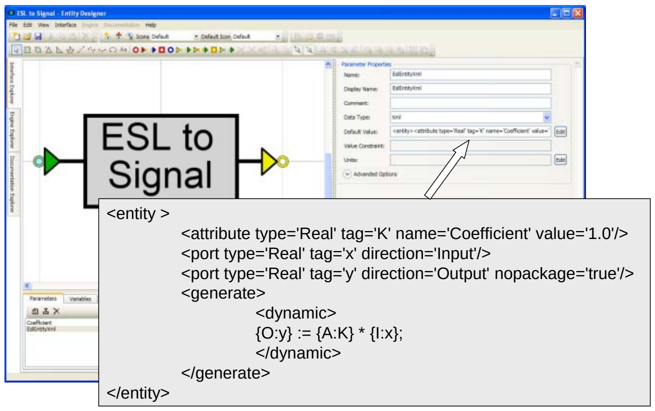
VTB 2011 comprises *Entity Designer* which allows a user to create new simulation entities (primary simulation elements from which a diagram can be constructed) with their ports and *Schematic Designer*, which lets a user construct a diagram of simulation entities, connect their ports and from which the simulation can be run. Within VTB there are several standard *Solvers* for handling different types of entity – Signal Solver, Natural Solver, DAE Solver, and Phasor Solver etc. Developers can add their own solvers by creating a .NET assembly that supports the appropriate VTB interfaces. The VTB framework will load the assembly into the Entity and Schematic Designer facilities, and invoke relevant code within it.

ISIM has developed an *ESL Solver* assembly for incorporation into VTB, building on a template/example project provided by USC.

- This supports the input and output ports that can connect ESL entities (these may be used to create user defined extension entities in the Entity Designer).
- It provides a set of built-in entities (equivalent to the standard set of ESL simulation elements).
- It has code that will be invoked when the user starts a run of the simulation in the Schematic Designer (*Run code*).
- It has code to handle each step of the simulation (*Step code*).
- There is also code to handle changes of simulation parameters at a pause in a running simulation, and in the engines of the built-in entities to support changes in parameters.

4.2. ESL entity creation

New VTB entities are created in Entity Designer where the appearance of the entity; its input and output ports; parameters and associated engine within the solver are specified. ESL entities have their own unique I/O port types ensuring they can only be connected to like entities. For the standard built-in entities, ESL code associated with an entity is identified by a special *EslIseClass* parameter which relates to a fragment of built-in XML code, which acts as a template (see XML fragment in Figure 4). To provide the ability to extend the set of ESL entities, a user may specify the associated XML code directly through an *EslEntityXml* parameter as shown in Figure 3



```
<entity >
        <attribute type='Real' tag='K' name='Coefficient' value='1.0'/>
        <port type='Real' tag='x' direction='Input'/>
        <port type='Real' tag='y' direction='Output' nopackage='true'/>
        <generate>
                <dynamic>
                {O:y} := {A:K} * {I:x};
                </dynamic>
        </generate>
</entity>
```

Figure 3 Setting the EslEntityXml parameter

```
<GenerateEsl>
<entity EslIseClass='appAbsoluteAbs'>
      <port type='Real' tag='x' direction='Input'/>
      <port type='Real' tag='y' direction='Output'
nopackage='true'/>
      <generate>
                  <include>absx</include>
                  <dynamic>
                  {O:y} := ABSX({I:x});
                  </dynamic>
      </generate>
</entity>
<entity EslIseClass='appAcosineAbs'>
      <port type='Real' tag='x' direction='Input'/>
      <port type='Real' tag='y' direction='Output'
nopackage='true'/>
      <generate>
                  <dynamic>
                  {O:y} := ACOS({I:x});
                  </dynamic>
      </generate>
</entity>
```

Figure 4 Fragment of code generation XML

4.3. Run code

When the user invokes a run of a simulation in the Schematic Designer, the ESL Solver invokes a singleton EslProgram object to handle it. This goes through the following sequence (if no errors are encountered):

- In the *Gather* phase it runs through the entities in the simulation diagram to identify entities which it can handle.
- It identifies entities which have an *EslIseClass* parameter as built-in entities, and those with a *EslEntityXml* parameter as extension entities that have been developed according to the required conventions.
- For built-in entities it obtains an XML fragment from the built-in source that defines the required ESL code.
- For extension entities it uses the XML fragment from the *EslEntityXml* parameter value for this purpose.
- Next it analyses the entity, and creates a mapping between the attributes specified in the XML fragment and VTB parameter, and maps the input and output ports specified in the XML with VTB ports. It also flags up whether the VTB parameter is allowed to change at runtime and whether a VTB port is specified for output, noting these as needing to be set appropriately in a simulation step.
- Having constructed the representation of ESL entities in the simulation diagram, in the *Generate* phase, it generates ESL code making use of the templates in the XML fragment for each entity. This code is written as a temporary file (*EslGenerated.esl*). The code for the ESL model is expressed as an ESL embedded segment form so it will be able to be used in .NET. (Other ways of incorporating the ESL model have been considered,

but this is currently viewed as the most efficacious approach).

- Next, in the *Run* phase, the standard ESL eslgen facility is invoked, which creates a .NET assembly (*EslGenerated_clr.dll*) representing the ESL model for the schematic diagram.
- Because this is changed for every fresh run of the simulation, this assembly cannot be directly loaded by the ESL Solver. Instead a proxy for the ESL model is used, and a separate assembly (*EslModelAccess.dll*) which runs in a separate .NET *AppDomain* (which is unloaded at the start of this phase) wraps the ESL model proxy which can load the ESL model assembly.
- Next some *look up* information for the ESL variables who's values may need to be input or output into the VTB framework is obtained, for efficiency in a simulation step.
- Then the ESL model (via the proxy) is initialised and made ready for the first simulation step.

4.4. Step code

When the VTB framework moves the simulation forward a step, the ESL Solver uses the EslProgram object to perform the advance of the simulation (it also does this at the beginning of a simulation). This code does the following:

- It obtains the values ESL model variables that have been designated as needing to be set in a step (via the proxy), and sets the corresponding VTB port values.
- Then it invokes the ESL model (via the proxy) to run forward to the next ESL *communication interval* (corresponds to the VTB step).

The sequence of operations is shown in Figure 5 and Figure 6.

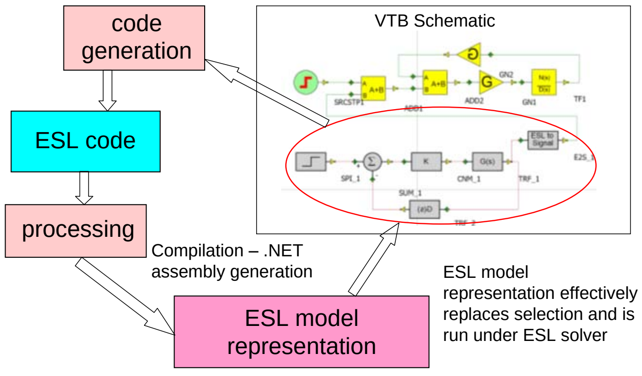

code generation
ESL code
processing
Compilation – .NET assembly generation
ESL model representation
VTB Schematic
ESL model representation effectively replaces selection and is run under ESL solver

Figure 5 Integration cycle

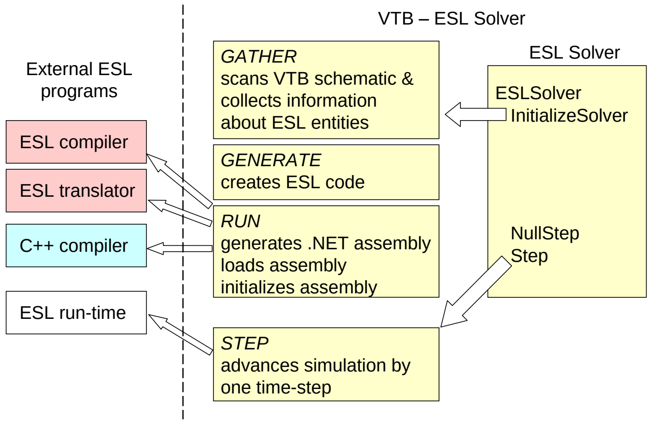


Figure 6 Implementation details

5. SOME RESULTS

An example of a VTB schematic for a servo system, consisting entirely of ESL entities, is shown in Figure 7.

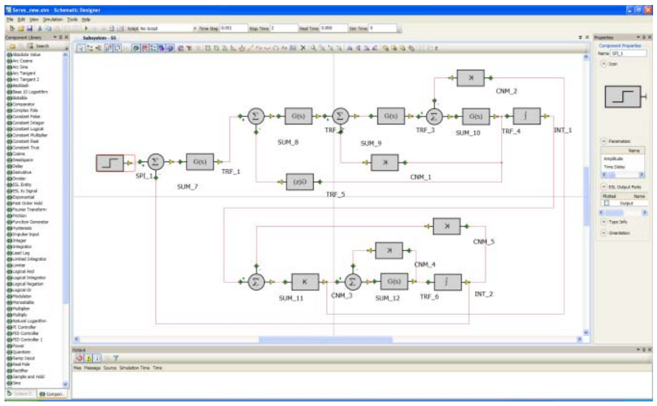


Figure 7 ESL in VTB schematic for Servo System

Figure 8 shows graphical results for the system compared with a corresponding plot generated in ESL.

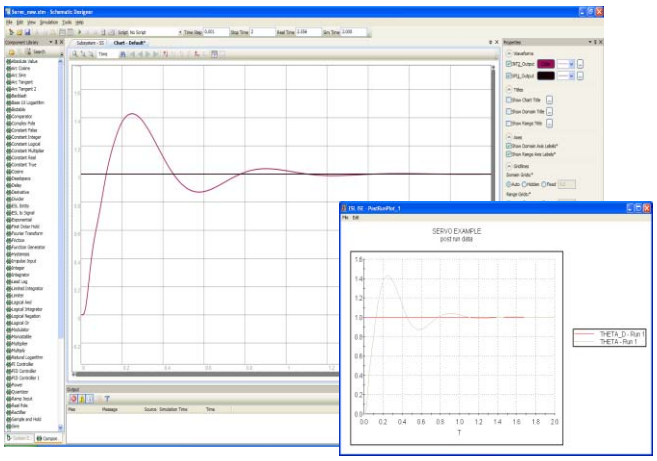


Figure 8 VTB and ESL Servo System results

6. THE NEXT STAGE

The primary objectives stated in section 3 have been achieved. The next stage is to replicate full ESL functionality. This entails:

- Allow the user to specify part of an ESL simulation as ESL textual code. Ideally a user would be able to open a text editor from the VTB and enter code in the form of an ESL submodel (as is the case with ISE). When the text editor is closed, the submodel would appear in the list of ESL components (or in a separate list) ready for inclusion in an ESL schematic. *This feature, in some form, is considered essential as it gives the user access to the full power of the ESL language*.
- Allow a user to import submodels from an external ESL library directory. Such imported submodels would then appear as ESL components in the Component Library. As with above, this implies the automatic generation of a default icon for the submodel.
- Allow the user to designate an ESL schematic (or a selection of an ESL schematic) as a submodel. The newly specified submodel would be available from the list of ESL components. It is possible this could be achieved through the VTB sub-system concept.
- 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 *coupling* components having both ESL and Signal ports.

7. LESSONS LEARNED

The primary objective of this on-going research project is to look at ways of improving the ability to work with widely differing simulation tools in an efficient and productive manner. The VTB was designed with this objective specifically in mind and it is not surprising therefore that the integration of a third-party product, ESL, has been relatively straight-forward so far. The ease with which such integration can be implemented depends largely on the availability of good well defined interfaces in the host software and a flexible modular structure in the integrating package. This was the case with the VTB and ESL software. The aim in this particular phase of the development was to retain the specific look and feel of ESL while taking advantage of features of the host software not offered in ESL (3D animation and access to an extensive range of native simulation entities, for example). A good working approach has been established and ideas such as the use of XML templates for code generation and the use, where ever possible, of existing external programs (ESL compiler, translator and eslgen utility) have proved successful. The fact that the original developers of ESL were responsible for the integration has undoubtedly eased the task and highlights the need for a close relationship between the developers of both the host product and the product being integrated. While all simulation tools are different, it is believed that the approaches and techniques used in this exercise could be considered if the integration of other third-party simulation tools was anticipated.

Whilst at this stage of the development it is difficult to quantify the improvements in efficiency and effectiveness of the simulation process that may result from the work described here, it is believed that such improvements will result from the following:

- The user has access, from a common environment, to the functionality of a range of alternative simulation tools, each offering particular features and advantages.
- When accessing a particular tool, the appearance of basic simulation components and diagrammatic representation closely resembles those of the native package.
- As the project develops, integration of the portions of the simulation created using different tools will be automatically taken managed by the host package.

8. CONCLUSION

Large-scale simulations require the use of widely differing modelling tools. Problems associated with the integration of such tools have been examined and a particular case study of the integration of ESL and the VTB has been described. The result of this integration is to allow the appearance and functionality of one simulation tool to be replicated within another, giving the user access to a wider range of options within a single environment.

Acknowledgements

This research has been supported by the Office of Naval Research through Award No. N00014-10-1-0625. The author also acknowledges 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.