ESL-ISE - A SIMULATION TOOL DEVELOPED FOR THE SPACE INDUSTRY

John Pearce
ISIM International Simulation Ltd, 26/28 Leslie Hough Way
Salford, M6 6AJ, UK, e-mail isim@cogsys.com
Roy Crosbie
California State University, Chico CA 95929-0003, USA
crosbie@ecst.csuchico.edu

Keywords: Graphical packages; Interactive programs; Continuous simulation

ABSTRACT

ESL-ISE (European Space Agency Simulation Language - Integrated Simulation Environment) is an advanced simulation tool, originally developed for the space industry, but which finds application in any field where complex non-linear dynamic system simulation is required.

A summary is presented which outlines the development of the tool through a series of ESA funded contracts and identifies key space related applications. The architecture and major features of the underlying simulation language (ESL) are described.

The newly developed graphical user interface, ISE, is described. ISE provides a powerful graphical editor enabling systems to be described in block diagram form while allowing the inclusion of ESL code modules where appropriate. A toolbox concept allows a simulation element palette to be configured for specific application fields. The ISE environment supports simulation program development, interactive control of simulation execution, and post-run analysis. During simulation execution, the user has access to all simulation program variables and parameters, which may be examined and changed. Run-time and post-run display of results (in both graphical and numerical form) is managed through a comprehensive display manager.

A number of the features of ESL and ISE are illustrated through an example of an ESL program and an ISE application

BACKGROUND

The simulation package, ESL-ISE, has evolved from a series of contracts undertaken by ISIM International Simulation Limited, and the University of Salford for the European Space Agency over a period of some twenty years. The initial contract (Hay et al 1981), which was a research study of simulation algorithms suitable for parallel architecture hardware, produced a proposal for a new simulation language standard (Crosbie and Hay 1982) - CSSL81. The proposed standard included features that allowed the decomposition of a large simulation into segments that could, in principle, be executed on parallel hardware. (Although initially this feature was emulated on a single processor, distributed simulation over a network of computers was fully implemented at a later date).

A second contract saw the implementation of a minimal software suite, which supported the proposed CSSL standard and the European Space Agency Simulation Language (ESL) was born. There then followed a series of contracts in which the language was extended and enhanced to meet the requirements of ESA. These extensions included the addition of graphical interfaces for block diagram model description, interactive simulation execution control and graphical analysis of results. Other enhancements included embedded and remote simulation capability, C++ translation, matrix arithmetic and both single and multiple-variable transfer function representation of dynamic elements.

During recent stages of the product development, a completely new graphical user interface has been added providing an Integrated Simulation Environment (ISE) from which all phases of the simulation process can be managed.

Although initially developed for the European Space Agency, ESL-ISE is a general-purpose continuous system simulation tool, with discrete event capability, that finds applications in the non-space sector as well as the space sector. The following are a selection of past and recent applications:

- Design of the Giotto (Halley's Comet probe) "Despin" antenna system.

- Investigation of thermal vibrations in the solar panels of the Hubble Space Telescope. (Poelaert 1987)
- Modelling of Nickel Cadmium battery systems for ERS1 (Earth Resource Satellite). (Hay 1987)
- Attitude Control Computer's Environment Simulation (ACC EnvSim) for the XMM Software Validation Facility (SVF). (Holiday 2000)
- Dynamics Simulation Library of the Model Library for Software Validation (MOLISOVA) developed by GMV.(Bonillo 2000)
- Software Validation Facility for the Attitude and Orbit Control Subsystem (AOCS) software of the scientific satellite ISO and the integrated data handling and AOCS software for communications satellite Artemis (Aidt and Mejnertsen 2000)
- Off-shore Gas-Rig training simulator.
- Gas Compressor station simulation (Kraft and Pearce 2000)
- Water filter bed simulation.

ESA SIMULATION LANGUAGE (ESL)

ESL is an powerful continuous system simulation language. Developed originally for the European Space Agency for use on advanced space projects it is now widely used in industry and academia for applications requiring *accurate* and *robust* computer simulations of dynamic systems. ESL has the following features:

ESL Features

- robustness (a simulation engine that runs for ever)
- handling of very large models
- extensive checking of model correctness
- accurate treatment of discontinuities
- wide range of numerical integration algorithms including stiff algorithms
- vector and matrix arithmetic
- both differential equation and transfer function model description
- submodel concept allowing hierarchical modelling of complex systems
- distributed simulation over a number of computers
- interpreter mode or translation into FORTRAN or C++
- embedded simulation and interface to other programs
- snapshot facility allowing the state of a simulation at a specific time to be saved for resumption later
- real-time capability

The ESL software

The ESL software suite, shown in Figure 1, comprises three main programs - the *compiler* which converts an ESL source program into an intermediate code (h-code); an *interpreter* which executes the h-code directly; and a *translator* which converts the h-code into either C++ or FORTRAN which is then further compiled and linked with the appropriate ESL run-time library and any user libraries to produce an executable program. A user has the option of using the interpreter route for fast program development or the translator route for efficient production runs and the ability to link to external libraries.

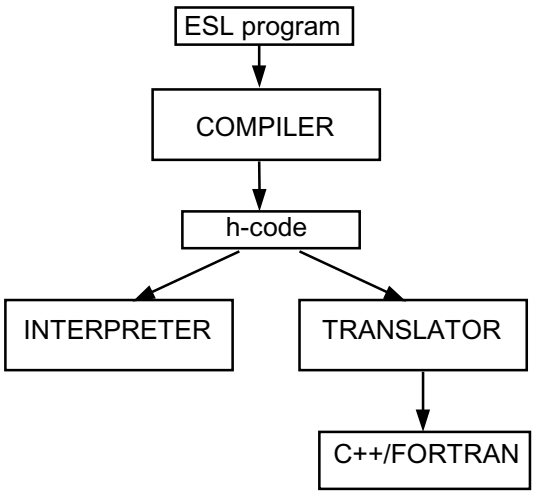

Figure 1

ESL Program Structure

ESL has a modular structure allowing a complex system to be modelled in an hierarchical manner. The highest level module of the dynamic description is the **model** which can call lower level **submodels**. A single submodel may be called more than once, each call representing a separate instance of the submodel (this illustrates the object oriented nature of an ESL program which maps naturally in to C++ when the translator option is chosen). Other modules permitted in ESL include: **procedure** containing non-dynamic code; **packages** which allow sharing of data between modules; and **segments** which are to be executed concurrently. At a level above the model is an experiment which defines how the simulation is to be performed. Figure 2 shows the general structure of an ESL program.

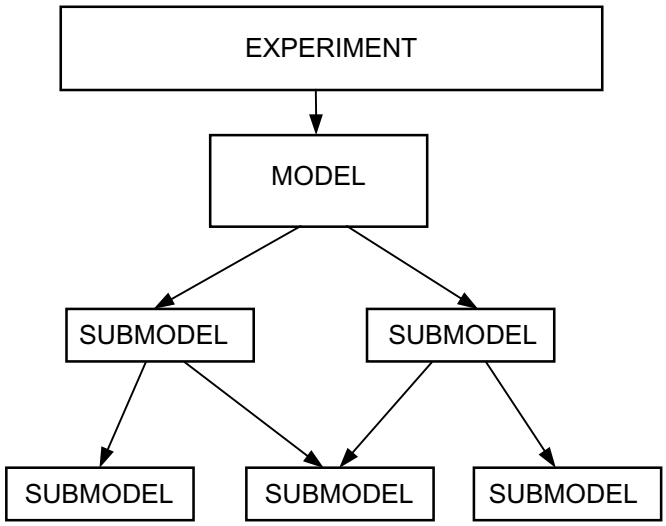

Figure 2

Simple ESL Example

To give a flavour of the ESL language, a simple example is presented in presented in Figure 3. The program simulates the vertical flight of a rocket governed by the equation:

$$height'' = \frac{thrust - drag}{mass} - g\ ,$$

where *thrust* is constant provided fuel is available, *drag* is proportional to velocity squared and *g* is acceleration due to gravity.

In this example we have a single model and an experiment. Notice the traditional INITIAL-DYNAMIC-TERMINAL structure of the model. Code in the STEP region is executed every integration step while the COMMUNICATION region is executed at regular time intervals as defined by the parameter "cint". The experiment makes several call of the model for different initial fuel quantities. Notice the use of the "if" and "when" structures to detect the exhaustion of the fuel supply and the rocket maximum height.

```
STUDY
 MODEL rocket(REAL: max_ht := REAL: f0);
  REAL: height, drag, thrust, velocity,
        fuel, Mass;
  CONSTANT REAL: g/9.81/, Mrk/300.0/,
                 burn/20.0/;
  LOGICAL: power, done/false/;
  INITIAL
--Initial conditions.
   height:=0.0; height':=0.0; fuel:=f0;
   max_ht:=0.0;
  DYNAMIC
-- Rocket dynamics.
   velocity:= height';
   drag:= 0.5 * velocity * abs(velocity);
   Mass:= Mrk + fuel;
-- Do we still have fuel and hence power?
   power:= fuel > 0.0;
-- Thrust is constant until fuel exhausted.
   thrust:= IF power THEN 35000.0 ELSE 0.0;
   height'':= (thrust - drag)/Mass - g;
   fuel':= if power then -burn else 0.0;
-- Detect maximum height.
   WHEN height' < 0.0 then
    done:= true;
    max_ht:= height;
   END_WHEN;
   STEP
-- Save results for post run analysis.
    PREPARE "rocket", t, height, velocity,
            thrust, fuel, drag, Mass;
-- Stop when rocket starts decent.
    TERMINATE done;
   COMMUNICATION
    Tabulate t, height, velocity;
   TERMINAL
    print "End of run";
END ROCKET;
-- EXPERIMENT
 REAL: f0, max_ht;
-- Set integration parameters
 algo:= rk5;  cint:= 15.0; tfin:= 120.0;
-- Do simulation for varying initial fuel.
 FOR f0:= 1400.00 .. 2000.0 STEP 200.0
 LOOP
-- Call the model to do a simulation run.
  rocket(max_ht := f0);
  print "With fuel ", f0:6.1, " kg",
  " height achieved was ",max_ht:-10.1,"m.";
 END_LOOP;
END_STUDY
```

Figure 3

INTEGRATED SIMULATION ENVIRONMENT (ISE)

At an early stage in the evolution of the ESL software suite, there existed a number of independent programs - ESL compiler; ESL Interpreter; FORTRAN and C++ translators; Graphical block diagram input program (IMP); Post run graphical display program (DISP); ESL Simulation Execution Control (ESL SEC). There was a degree of integration. For example, in the non-ESL SEC environment, the compilation, interpretation or translation and execution processes could be initiated from the graphical input program (IMP). Run time and post-run graphical results could also be specified from IMP (the latter by invoking DISP). Under ESL SEC control (at the time only available on Sun Solaris systems), all the simulation processes could be invoked from the ESL SEC user interface. However, although innovative at its time of creation, IMP had limitations and used non-standard window management (on PC systems, it was a DOS program). Also, ESL SEC was written in the OPEN LOOK

style, by then outdated. It was proposed to re-write the graphical user interfaces using Motif and bring the various components of ESL together within an Integrated Simulation Environment (ISE) shown schematically in Figure 4. ISE provides the following features:

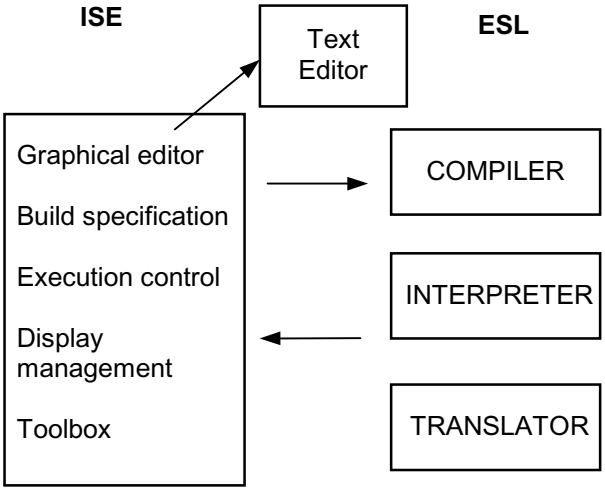

Figure 4

- multi-window graphical block diagram editor for model construction
- inclusion of ESL coded submodels where appropriate
- interactive control of simulation execution
- run-time and post-run graph plotting
- user configurable simulation element palette
- application specific toolbox capability
- display manager
- on-line help

The functionality of the old IMP, DISP and ESL SEC programs have been incorporated into ISE, which now provides the development environment from which each stage of the simulation activity can be managed.

ISE includes a graphical editor for block diagram style model descriptions, while allowing textual ESL code to be used where appropriate (for example, to describe highly non-linear elements). Standard simulation elements can be selected from a palette and interconnected on a canvas to build up the simulation description.

ESL submodels can be created and included in a diagram through a special submodel element. An option is provided to configure the simulation element palette itself with user defined submodels and thus create specialised application specific *toolboxes*.

Once a simulation program has been created (graphically, textually or a combination of both), compilation is initiated from ISE. The user then has the option to execute the program immediately through an interpreter, or further translate it to FORTRAN or C++. The resulting executable program may then be run from ISE. In either case, execution is managed by an interactive control panel, which provides run-time control of the simulation.

Access is provided to all program variables and parameters from the control panel. This includes simulation parameters such as the communication interval, final simulation time, choice of integration algorithm and error tolerances. All variables and parameters can be set and changed dynamically from the control panel.

Graphical and tabulated output can be specified on the block diagram through the use of special simulation display elements or alternatively from a versatile display manager window.

All run time commands and output specifications can be logged to a driver file that can be used at a later time to repeat simulation scenarios and full support is provided for the ESL snapshot facility.

AN EXAMPLE - A SERVO CONTROL SYSTEM

As an example if the ISE interface, we consider a simulation of the servo control system shown in Figure 4.

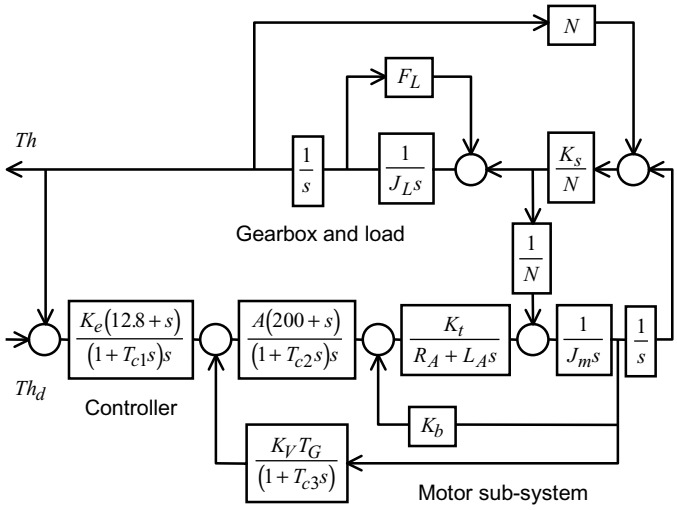

Figure 5

The system naturally splits into three sub-modules - *the Controller*, *the Motor sub-system* and *the Gearbox and load*. These modules are represented by ESL submodels interconnected at the model level. Figures 6,7 and 8 are the ISE diagrams for the three submodels. Note that the transfer function annotation on the diagrams is generated automatically from the definition of the simulation element attributes. Figure 9 shows the complete ISE window after the simulation has run with graphical and tabulated output displayed on the screen.

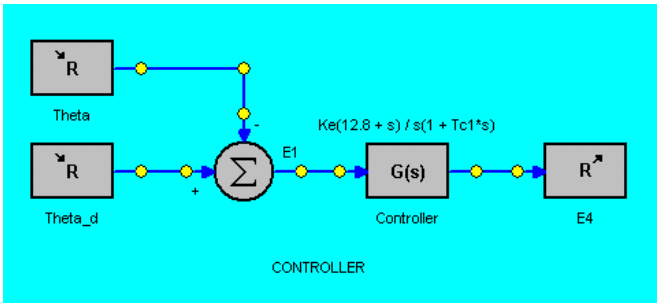

Figure 6 - The Controller

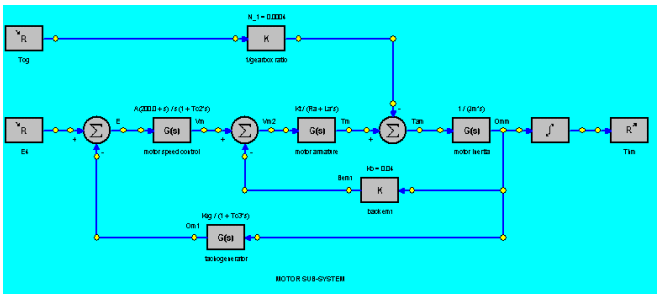

Figure 7 - The Motor Sub-System

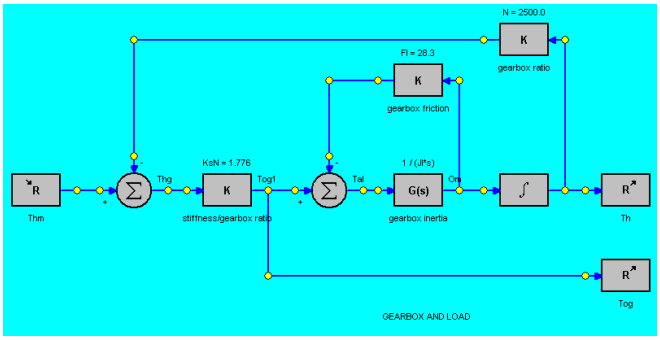

Figure 8 - The Gearbox and Load

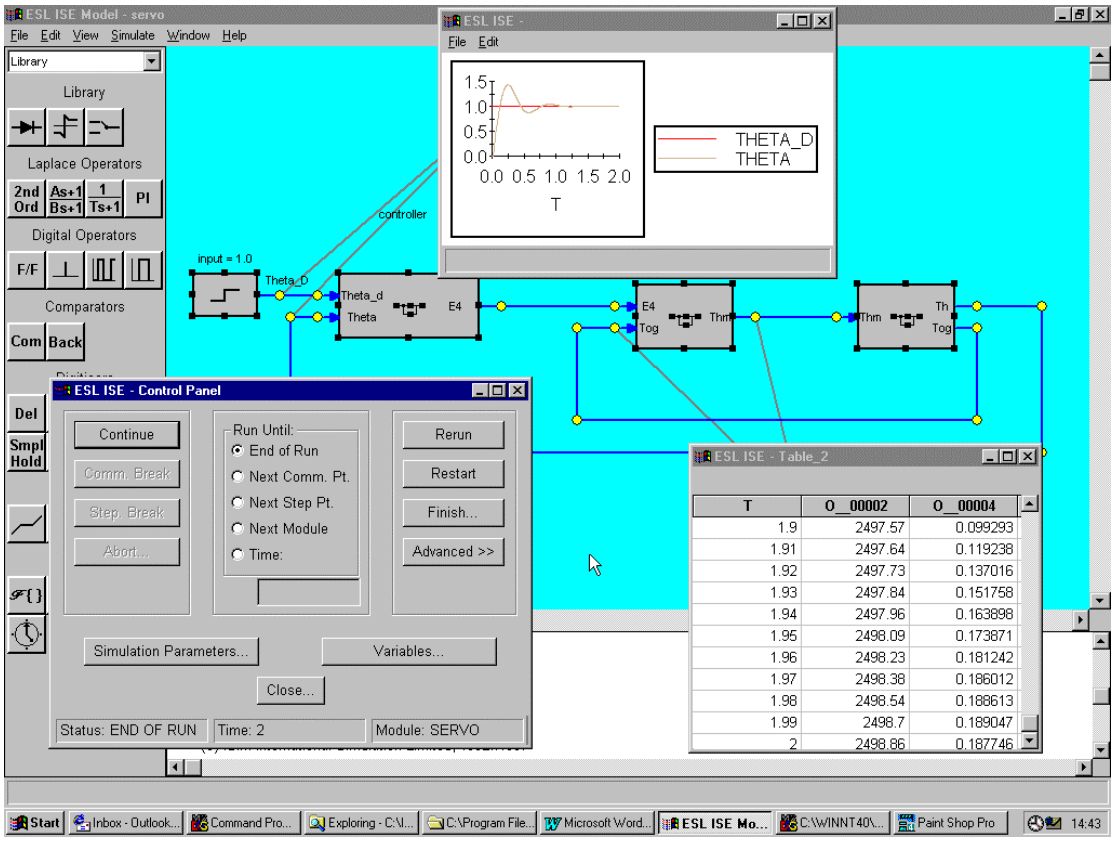

Figure 9 - Screen dump of Servo example

THE FUTURE

In continuing to develop the ESL-ISE product, the following are some possibilities being considered:

- Vector diagrams - currently block diagrams only allow scalar connections between simulation elements, i.e. a line on the diagram represents a scalar variable. A logical extension would be to allow vector connections between a set of new multivariable simulation elements thus taking full advantage of ESL's vector and matrix capability.
- Combined continuous discrete (CCD) simulation - A study undertaken for ESA established the capability of ESL for CCD simulation and recommendations were made for the extension of the language to facilitate this. Such language enhancements could be mapped through to the graphical interface.
- Component objects - the contract has shown that an ESL simulation can be distributed over a number of platforms and other projects have integrated ESL simulations with other applications. A related approach would be to provide a means of automatically creating ESL simulations as COM or CORBA objects, thus facilitating communication with other programs and the ability to link to Web pages.
- Specialised toolkits - the ISE simulation element palette can be configured with user defined submodels. A possible area for development would be the creation of libraries submodels for specialised applications. These would be mapped onto appropriate simulation elements on the palette.

CONCLUSION

The development and current state of the ESL-ISE simulation tool has been described. The product represents the results of several contracts for the European Space Agency carried out over a number of years and has now reached a state of maturity making it suitable for both space and non space applications.

TRIBUTE

Many people and organisations have been instrumental in the creation of the ESL-ISE simulation software. However, this paper would be incomplete without special mention of driving force behind the ESL-ISE project - the late Professor John Lewis Hay - founder of ISIM International Simulation Limited. It is through John's knowledge of needs of industry and commerce for robust and accurate simulation software and his great attention to detail that the product is and continues to be successful.

ACKNOWLEDGMENTS

The work described was performed under ESTEC contracts: 4155/79, 4790/81, 5663/83, 6466/85, 9041/90/NL/JG, 1011/92/NL/JG. The authors gratefully acknowledge the services provided by the University of Salford and Salford University Business Services Ltd. and the essential contribution of Cogsys Limited, Salford, UK.

REFERENCES

Bonillo, C., Vega, E, and Mejnertsen, S. 2000. "Flight Dynamic System Modelling using ESL". In *Proceedings of 2000 Western Multiconference* (San Diego, CA, Jan 23-27). The Society for Computer Simulation International, San Diego, CA, USA.

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.

Crosbie, R. E. and Hay, J. L. 1982. "Towards New Standards for Continuous-System Simulation Languages". In *Proceedings of Summer Computer Simulation Conference* (Denver, CO, USA, July).

Hay, J. L. 1987. "ESL Simulation of Spacecraft Battery Cells". In *Proceedings of UKSC Conference on Computer Simulation* (Bangor, UK, Sept 9-11). UKSC/SCS, Ghent, Belgium, 92-97.

Holliday, P. 2000 "XMM Attitude Control Computer Environment Dynamics Simulation for SVF". In *Proceedings of 2000 Western Multiconference* (San Diego, CA, Jan 23-27). The Society for Computer Simulation International, San Diego, CA, USA.

Kraft, R. J. and Pearce, J. G. 2000. "Using ESL in an Integrated Real-Time Compressor Simulation Application". In *Proceedings of 2000 Western Multiconference* (San Diego, CA, Jan 23-27). The Society for Computer Simulation International, San Diego, CA, USA

Poelaert, D. 1987. "Hubble Space Telescope Solar Array: A Thermally induced Disturbance Torque". Technical Report ESA STM-238, ESTEC, Noordwijk, The Netherlands.