Real Time Distributed Simulations Using SIMSAT 4.3

Joshua Whitty
Terma GmbH, Europaplatz 5, 64293 Darmstadt, Germany
jowh@terma.com

Abstract - **The need for large-scale real-time simulations is growing in demand within many diverse industries. This places ever-increasing performance requirements on the hardware platforms required to run the simulations. Possible solutions to this include the use of multiple core computers and a network of computers to distribute the simulator processing load across many cores and machines.**

This paper examines some recent examples of such simulations within the European Space sector, and how SIMSAT 4.3 simulation infrastructure has been extended to satisfy the requirements to run distributed real-time simulations.

The underlying simulation environment used by the European Space Agency (ESA) and the European Space Operations Centre (ESOC) for spacecraft simulators since the early 1990's is the SIMSAT real time infrastructure. SIMSAT originally ran under VMS, but since then it has been ported to many different Operating Systems, i.e. Windows NT/2000/XP and then most recently SuSE Linux Enterprise Server 9 and 11. It provides the real-time environment for spacecraft simulators to verify the ground segment readiness prior to launch, including the validation of the mission control system, validation of spacecraft procedures and the training of the spacecraft operators. The requirement to develop simulations for large and complex space systems within the European Space Market, for example Galileo, places heavy demands on the simulation platform. This increasing performance demand has driven the development of a distributed version of the SIMSAT infrastructure.

The paper introduces SIMSAT at a conceptual level, the current SIMSAT Linux 4.3 Software Architecture, and the benefits of its framework "plug-in" design which make it so versatile, flexible and extendable. This plug-in framework allows fast and rapid development of new simulation components that plug-in to the SIMSAT framework, for example, to support the distribution and running of a simulator across a distributed network, or to selectively load various models and assemble different simulated components to run.

The system, network performance, time synchronisation, and configuration requirements of a real-time distributed simulation system are identified. The paper demonstrates how these constraints were solved through the software architectural design principles applied.

The results show that the SIMSAT 4.3 Infrastructure Software meets the system and performance requirements of running distributed models across a dedicated network. The new SIMSAT can be utilised across diverse industries to run large-scale distributed simulations.

SIMSAT is an ESA product for which holds the full IPR (intellectual Property Right).

I. INTRODUCTION

With the advances in technology within the European Space sector the demands for larger simulation systems have increased. This places increased performance requirements on infrastructure software to ensure that the simulations maintain real-time and that configuration control of each element within the simulation is easily maintained. The processing power of a single machine may not be enough for a large-scale simulations. This has driven us to design a parallel distributed simulation system to distribute the processing load of each element in the simulation.

The ESA Galileo project is a large and complex system to simulate requiring the simulation of many elements in parallel. Two examples of this are the Galileo Constellation Operational Simulator (CSIM) and the Assembly Integration and Verification Platform (AIVP). Both of these projects have many elements that are simulated in parallel with high processing demands that far exceed the capabilities of a single machine. ESOC operational simulators are also following the same trend, with the SWARM constellation simulator, and SENTINEL-1 requiring more than one spacecraft to be simulated. Due to this, the SIMSAT 4.3 infrastructure (ESA's Real time Operations Simulations Infrastructure) has been extended to support the running of simulations across multiple cores, CPUs and also a network.

The Galileo Constellation Operational Simulator simulates multiple spacecraft in parallel. Its purpose is to simulate multiple spacecraft to verify the ground segment readiness prior to launch, including the validation of the mission control system, validation of spacecraft procedures and the training of the spacecraft operators.

AIVP is the verification platform for the Galileo Ground Segment. It simulates many different Ground Stations, for example, Galileo Sensor Station (GSS), Message Generation Facility (MGF), the Orbitography and Synchronisation Processing Facility (OSPF), as well as the Real-Time Network Emulator that simulates the network and passing of messages between all Ground Stations. The purpose of the simulator is to test the Galileo Ground Segment as a whole and to replace the software models with the real stations and hardware when they become available to verify and validate the system.

SWARM is ESOC's first constellation simulator, modelling three spacecraft, using SIMSAT 4. The primary objective of SWARM, through its constellation of 3 satellites, is to provide a survey of the Earth’s geomagnetic field and provide the first global representation of the field’s variations over time. This project inherited the distributed components developed for the CSIM project on SIMSAT 3. ESOC recognised that the demand for distributed simulations were increasing and decided to add generic distributed components to SIMSAT 4.3 that would work for all existing and future constellation simulators.

II. PARALLEL/DISTRIBUTED SIMULATIONS

Parallel/distributed simulation technology has been around since the early 1980s with initial advancements arising from the Defence Community to simulate military scenarios to train soldiers. Since then it has made great advancements and is now widely used in the Gaming Industry and the Internet. The technology enables a simulation program to be executed in parallel on distributed interconnected computers across a network. These machines may be geographically distributed across a building or even the world. There are primarily two main benefits to this.

1. *Reduced execution time* – With multiple machines and thus multiple processors, the processing load is divided and shared amongst each processor to reduce the load of the system processing to ensure that real-time is maintained.
2. *Geographical Distribution* – The machines are connected to a network and therefore are not geographically constrained.

Parallel/distributed simulation technology is made possible by the advancement of three essential technologies:

1. *Integrated Circuits* – An inexpensive computer which allows building up a network economically feasible.
2. *High Speed networks* – Whether connected to a Local Area Network (LAN) or the Internet enabling fast data transfer.
3. *Modelling and Simulation* – Technology to construct models of actual real-world systems that when run on a normal computer can emulate and be representative of the real system.

Due to these advancements in technology we were able to solve the problems associated with simulating these large-scale projects using a distributed SIMSAT system.

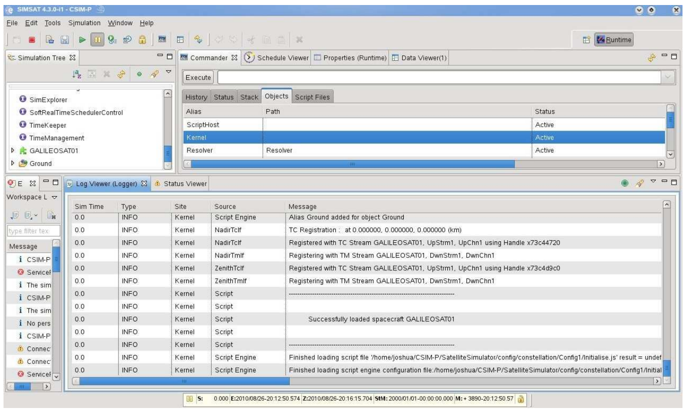
III. SIMSAT

SIMSAT (The Software Infrastructure for Modelling Satellites) is the real-time infrastructure software used by the European Space Agency (ESA) and the European Space Operations Centre (ESOC) to run simulations since the early 1990s. It was originally written to run on VMS, but over the years it has been ported to many different Operating Systems including Windows NT/2000/XP, SuSE Linux Enterprise Server (SLES) 9, and the latest being SLES 11. Up until now it has only been used for simulations that were capable of running on single machines, for example, single spacecraft simulations containing Ground models to verify the ground segment readiness prior to launch, including the validation of the mission control systems, validation of spacecraft procedures and the training of the spacecraft operators.

SIMSAT contains a Graphical User Interface (GUI) called the MMI (Man Machine Interface) to allow simulator runtime behaviour to be actively monitored and controlled. The MMI can display the status of the simulated object (i.e. a spacecraft) including all contained model parameters, services and variables. The simulation environment traps all exception conditions and displays error messages for all erroneous actions and anomalous behaviour. An on-line help facility is also provided as part of the MMI - to replace the SUM.

The simulation runtime environment functionality is summarised below:

- Graphical User Interface (GUI).
- Scheduling of simulation models, commands and Java scripts.
- Message logging.
- Recording of simulation data over time.
- Visualisation of model data.
- Control of the simulator via commands and scripts.
- Saving / restoring simulation states (Break pointing).



Alias	Path	Status
ScriptHost		Active
Kernel		Active
Resolver	Resolver	Active

Sim Time	Type	Site	Source	Message
0.0	INFO	Kernel	Script Engine	Alias Ground added for object Ground
0.0	INFO	Kernel	NadirTclf	TC Registration : at 0.000000, 0.000000, 0.000000 (km)
0.0	INFO	Kernel	NadirTclf	Registered with TC Stream GALILEOSAT01, UpStrm1, UpChn1 using Handle x73c44720
0.0	INFO	Kernel	NadirTmlf	Registering with TM Stream GALILEOSAT01, DwnStrm1, DwnChn1
0.0	INFO	Kernel	ZenithTclf	Registered with TC Stream GALILEOSAT01, UpStrm1, UpChn1 using Handle x73c4d9c0
0.0	INFO	Kernel	ZenithTmlf	Registering with TM Stream GALILEOSAT01, DwnStrm1, DwnChn1
0.0	INFO	Kernel	Script	--
0.0	INFO	Kernel	Script	
0.0	INFO	Kernel	Script	Successfully loaded spacecraft GALILEOSAT01
0.0	INFO	Kernel	Script	
0.0	INFO	Kernel	Script	--
0.0	INFO	Kernel	Script Engine	Finished loading script file '/home/joshua/CSIM-P/SatelliteSimulator/config/constellation/Config1/Initialise.js' result = undef
0.0	INFO	Kernel	Script Engine	Finished loading script engine configuration file:/home/joshua/CSIM-P/SatelliteSimulator/config/constellation/Config1/Initial

```
S: 0.000 E:2010/08/26-20:12:50.574 Z:2010/08/26-20:16:15.704 StM: 2000/01/01-00:00:00.000 M: + 3890-20:12:50.57
```

Fig. 1. SIMSAT 4.3 MMI GUI on SLES 11

The run-time Kernel is the heart of the simulation and is responsible for scheduling models, event logging, user commanding, visualisation of model data and the saving/restore of the simulation states. Here is a brief description of the SIMSAT components;

SimHost: The run-time process on the system that is the simulation. The SimHost always has one Kernel.

Kernel: The core object that contains all components within a simulation and allows control over the creation and contents of simulations.

Resolver: Exposes information of internal simulation objects to the outside world i.e. the MMI.

Explorer: Exposes all simulation model objects, data and services to the simulation. This is used in the MMI to display a hierarchical view of all objects in the simulation.

Logger: Allows the storage of messages produced by the simulation system for immediate or delayed review.

Storer: Allows the saving of simulation “states” to file on disk. This can then be reloaded at some later point to recreate the simulation as it existed when it was saved.

Scheduler: The scheduler stores a list of model events that represent the activity that will occur in the simulation future on its Schedule. The Scheduler also contains a **Time Keeper** which controls and monitors the passage of time in the simulation.

Data Server: Serves simulation data to the outside world i.e. MMI.

Recorder: Records simulated data to file over time. simulation.

Command Line Interface: Allows SIMSAT to be launched and commanded from the command line i.e. without the use of an MMI.

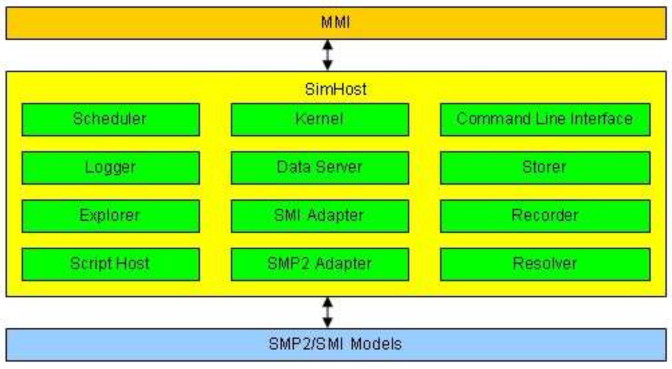


Fig. 2. SIMSAT 4.3 Non Distributed Top Level Architecture

One advantage of the SIMSAT 4.3 architecture, inherited from SIMSAT 3, is that all the components are based on CORBA technology. As such, these components are completely location independent in terms of their functionality and will continue to work in the remote case where they may be distributed across a network.

Another advantage is that SIMSAT 4.3 works in terms of Daemons, not machines. In order to start a simulation a Daemon must be running to act as a server to the client launching the simulation. In SIMSAT this client is the MMI, and the target Daemon spawns a new simulation host process (called SimHost). Since every machine on a local network can run one or many Daemons, and a Daemon can create one or many simulations, there is a great degree of flexibility when it comes to launching simulation executables on the same, or particular machines.

The current SIMSAT Kernel and MMI framework supports a “plug-in” architecture which makes it so versatile, flexible and extendable. This plug-in framework allows the development of new SIMSAT components that plug-in to the existing SIMSAT framework without altering or affecting the existing SIMSAT code base. By simply adding a file to the "components" directory of the SIMSAT installation, and entering the component information into the SIMSAT Kernel Architecture File, it can then be loaded into the simulation. The existing code base does not need to be recompiled. This allows us to develop specific components to support the running of SIMSAT across a distributed network, or specific components to meet the requirements of specific projects.

IV. DESIGN ISSUES AND SOLUTIONS

Corba and Network Latency

As mentioned above, SIMSAT components are based on CORBA technology. Even though this enables us to run the components on remote machines, there is a performance penalty to pay in terms of the amount of time taken to complete a single CORBA out of process call across a network.

A single CORBA call requires around 1 microsecond when the calling and called objects are located in the same process, but this can increase to around 50 microseconds when they exist in different processes on the same machine. This latency jumps to around 100-1000 microseconds when they are located on different machines, though this figure is highly variable and subject to the network configuration and conditions. This increase in the amount of time to complete one call leads to performance problems. Due to the high number of CORBA calls between the Kernel components and models, this has driven the design of the new SIMSAT Kernel components to allow distribution of parts (i.e. individual Models) of a simulation to other remote machines.

Distributing Kernel Components

To overcome the performance problems of running CORBA components on more cores, CPUs or remote machines and processes, some SIMSAT kernel components were distributed. The Architecture has one centralised “Master Kernel” per simulation which may contain many "Kernel Sites" that are installed on remote machines. Each Kernel Site acts as a local service provider, but provides default services from the Master Kernel if they are not available on the site.

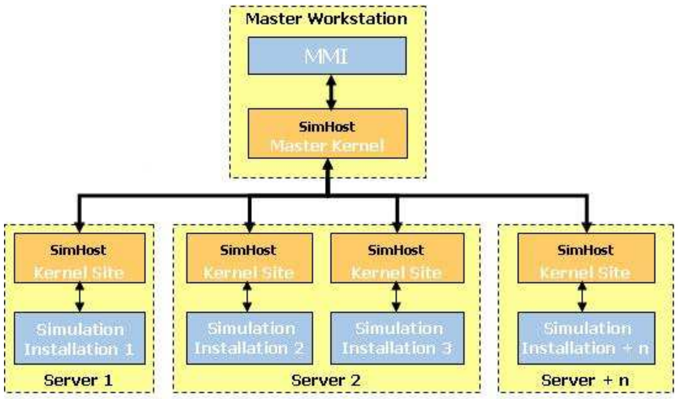


Fig. 3. Distributed SIMSAT Top Level Architecture

The Kernel, Storer, and Scheduler components have been redesigned to support distribution. Fig. 4 below shows the top level architecture of the distributed Kernel Master and Site components.

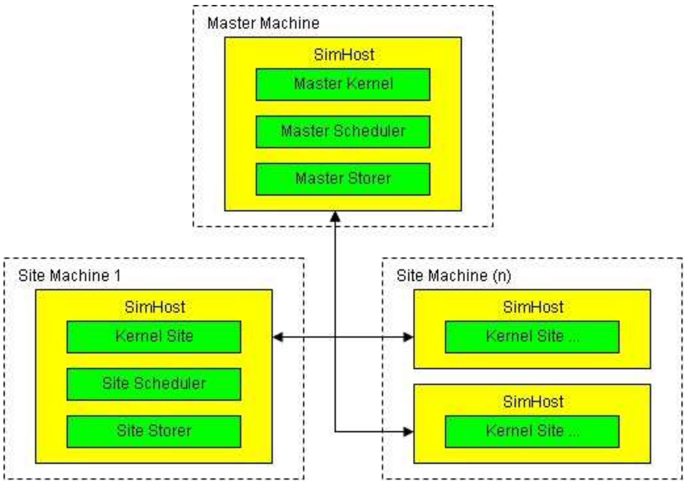


Fig. 4 Distributed SIMSAT Kernel Architecture

Note that SIMSAT 4.3 does not contain a distributed Logger as it is at present not foreseen as necessary. Other constellation simulators i.e. CSIM and AIVP, do contain distributed Loggers in their SIMSAT 3 infrastructure because they are required to simulate a large number of Kernel Sites, which is foreseen to produce a higher number of log messages. CSIM for example, will simulate 18 fully emulated spacecraft across a network.

Distributing the Storer

The total amount of time required to save and restore model data for a distributed simulation is greatly reduced by employing a parallel save/restore design using multiple threads. The design contains a Master Storer, located in the Master Kernel on the master machine, and a Site Storer located in the Kernel Site on the site machine. The Master Storer receives the instruction from the MMI and hands off this instruction on separate threads to each Site Storer, which save/restores the model's state to or from a local file.

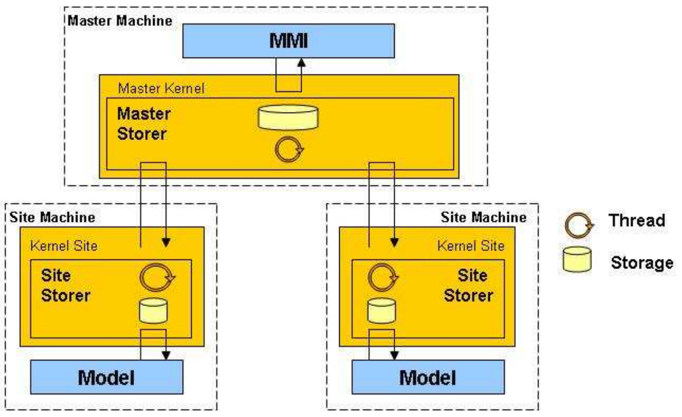


Fig. 5 Distributed SIMSAT Storer Architecture

Distributing the Scheduler Component and Time Synchronisation

The Scheduler component has the most amount of CORBA calls to make to other models within the simulation. It is not unusual for the Scheduler to execute 500 to 1000 events per second just for one element/spacecraft in the simulation. For this reason particular care was taken to reduce the number of Corba calls across the network in the design by having a Master Scheduler located in the Master Kernel on the master machine, and a Site Scheduler located in the Site Kernel on the site machine. With this design all models located on each site machine registers their events onto the local Site Scheduler. The Site Scheduler has its own schedule and executes all the local events locally. This eliminates all scheduled events/CORBA calls across the network. But this creates a new problem. How do we ensure that all Site Schedulers are synchronous with each other?

This design required a concept of being able to synchronise time within the whole simulation across all site machines as prototyping results showed that Site Schedulers very quickly ran out of sync with each other (to milli second level) just after a couple of seconds of running simulation time.

Two methods of time synchronisation were investigated and prototyped.

1. *Centralised, pull time service* – Each Site Scheduler requests the current time form a centralised time service (passive server algorithm)
2. *Centralised, push time service* – A central time service periodically sends new time values to the Site Schedulers

The SIMSAT scheduler is a "Soft Real-Time Scheduler". This means that the execution success of an event is more important than the event being executed at the precise time. Simulation time does not advance regularly like real-time because the scheduler advances simulation time forward when events are executed. Because of this a Centralised Push Time service would require more implementation effort as a mechanism to update simulation time to all sites to compensate for the latency from sending the time to the first Site Scheduler to the last Site Scheduler in the simulation is needed.

With a Centralised Pull Time service the Master Scheduler acted as the Time Keeper within the simulation i.e. one centralised controlled time. Each Site Scheduler requests the elapsed time at configurable periods, to synchronise its local scheduler. This reduces the number of network calls to a minimum. Each Site Scheduler has its own simulation time which is used to provide local models of the site with simulation time when requested

Every time the Master received a time synchronisation request, it stores and uses the Site Scheduler time to calculate the constellation simulation time. Two methods were introduced into the Master Scheduler to calculate constellation simulation time. One calculates constellation simulation time by using the slowest site, and the other mechanism uses the fastest site - which mechanism to use can be set by the user.

Because each site may have different events on the schedule a Centralised Pull Time method is better suited as each Site Scheduler would request time from the centralised time service at irregular intervals, and thus less effort is required in the implementation.

The Master Scheduler also contains a schedule. This is implemented to support the scheduling of Java scripts and commands which come from the MMI. It is not possible for Kernel Site Schedulers to add events to the Master Scheduler, as its simulation time is not used in the calculation of the constellation simulation time, as its events are not model driven.

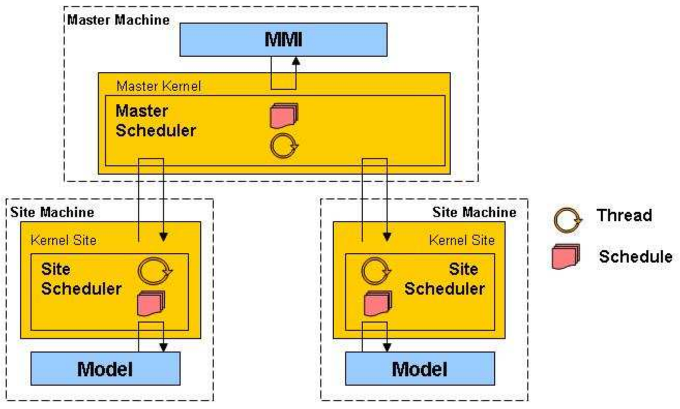


Fig. 6 Distributed SIMSAT Scheduler Architecture

Configuration Control

Even though the distributed aspects of a simulation are completely transparent to the user from the MMI, the management of configuration files for a distributed or constellation simulation across many computers can lead to real problems and confusion. Different simulators have different requirements and runtime environments. For example, the SWARM simulator, simulates 3 spacecraft on one computer. It does this by distributing each site SimHost on a separate core/CPU of one machine. Therefore a quad core machine is sufficient to run a constellation simulation, and no "real" distribution of the software across a network is required. Therefore only one set of configuration files on disk can be used for all 3 spacecraft instances. Other constellation simulators on the other hand have to simulate many more spacecraft i.e. CSIM, and this creates file synchronisation problems.

The approach taken on CSIM was to store all files in a centralized area on one machine, to avoid multiple copies of files spread over the network. Each site machine has a mounted drive to the centralized configurations area and can read and write to these folders. Each site simulation can therefore read and write to these files when required.

Shared Memory

Shared memory and code execution from multiple threads are protected using mutexes. This is an issue for the distributed components within the Master Kernel and any models that may communicate with each other.

MMI

The distribution of models is transparent to the users in the MMI. MMI plug-ins were developed to support the options for saving/restoring system level and element/site level “states” as well as aiding the users in building configurations. The SIMSAT 4.3 MMI Log Viewer can open multiple viewers and more filter options were added. This gives more flexibility to the users to see log messages from only one site, or one element within a site, or even log messages of only a particular type.

V. RESULTS

The aim of integrating the distributed components into SIMSAT 4.3 was to implement a generic solution that would work for current ESOC and ESA constellation distributed simulators, as well as ensuring that it would satisfy future requirements for distributed simulations. Since the SWARM project inherited the distributed components from CSIM, taking these components was a good starting point as they were already ported from SIMSAT 3 to version of SIMSAT 4.0.3. The CSIM project did have specific distribution requirements because it runs in a "true" distributed environment across a network of computers, that was not required for SWARM, so special attention was taken to reintroduce some missing CSIM functionality. The end result is an implementation that works for both of these simulators.

As part of the validation work to verify the success of a generic implementation in SIMSAT 4.3, both the SWARM and LEOP CSIM constellation simulators were ported to run on SIMSAT 4.3. All SWARM system tests were successfully ran and passed on site at ESOC. More effort was required to get the CSIM simulator ported to run on SIMSAT 4 because this simulator currently runs on SIMSAT 3, with its own Smp2Adapeter implementation. We were able to get CSIM running with the OBSW, without experiencing any real time slips. Because we ere not validating the simulator, but the distributed SIMSAT, no more effort was taken to get the system test load and run.

The SWARM simulator that simulates 3 spacecraft in parallel has been tested distributed. Using the ESOC ERC32 emulator, one spacecraft can be installed and run per computer, allowing two times real-time speed simulations. No real time slips were observed in the simulation. The CPU load of each SimHost on each machine lies around 30% and between 15 to 20% on the master machine.

The LEOP CSIM simulator uses more CPU, approximately 70% for each spacecraft running on its own SimHost. For this reason it could not be run faster than real time, but no real time slips were observed when running distributed.

Note that measurements were made on using a computer with a 2.8 GHz Intel processor and performance measurements are hardware dependent.

VI. Conclusion

SIMSAT 4.3 provides a powerful simulation infrastructure including features such as real-time access to all simulation models, hierarchal visualization of models, commanding of models, real-time data visualization and plots, real-time logging facility and scripting capabilities. Its flexible plug-in design allows developers to develop extra pluggable components that can be added to the software without modifying its code base and thus not requiring a rebuild of the source code. These plug-in components can be developed for specific project needs and can interact with the rest of the system through the provided interfaces.

The SIMSAT 4.3 to support distribution is a powerful addition to the real-time infrastructure software that can be used to simulate large-scale simulations. Based on the distribution of its internal components, the system distributes the processing of its elements to remote machines whilst maintaining synchronous time across the whole system. To the end user the distribution of the system is transparent, easily controlled and monitored through SIMSAT's MMI GUI.

It is recommended that one core, or CPU, is left spare per machine when running constellation simulations. This is because other applications on the computer require resources to run i.e. operating system, other applications etc, as well as Ground Models (TTC Streams). Another observation made is that when an operational simulator uses more than approximately 85% processing power per CPU, or CPU core, then real time slips and latencies occur in the simulation and it also puts more stress on the constellation.

The results from the Galileo Constellation Simulator and SWARM Simulator show that the distributed SIMSAT meets the demanding system performance requirements to run large-scale distributed models across a dedicated network. Using this distributed environment has clear benefits – not only in the space industry but across diverse industries where complex large scale distributed systems are required.

References

[1] J. Whitty, T. Baud, Fully Emulated Constellation, ESA, GAL-TN-SSL-CSIM-I-0002, Issue 1 Revision 0, 13.09.2006.

[2] J. Whitty, T. Baud, S. Straw, Galileo Constellation Simulator Architectural Design Volume 0, ESA, GAL-SDD-SSL-CSIM-A_0001 SDD Issue 1, Revision 0, 21.03.2007.

[3] Richard M. Fujimoto, Parallel and Distributed Simulation Systems, ISBN 0-471-18383-0.

[4] Galileo GMS AIVP – System Level Software Design Document, GAL-DD-VEG-AIVP-R_020078 Issue 1C, 19.01.2007