SIMBRIDGE: ADAPTATION TECHNOLOGY FOR SPACE MULTIDISCIPLINARY APPLICATIONS

Workshop on Simulation for European Space Programmes (SESP)

24-26 March 2015

ESA-ESTEC, Noordwijk, The Netherlands

Georgia Soulioti[(1)], Vagelis Doukakis[(1)], Nikolaos-Antonios Livanos[(1)]

[(1)]EMTECH

44, Kifisias Ave. Marousi, 151 25, Athens, Greece

georgia.soulioti@emtech.gr, vagelis.doukakis@emtech.gr, nikolaos.livanos@emtech.gr

ABSTRACT

For many years, efforts were carried out for standardization (SMP2, SSRA, REFA, etc.), and space models' integration (e.g. MOSAIC) in order to provide interoperability between systems. However, due to the interdisciplinarity of aerospace projects, the need for consolidation and adaptation in Simulation Environments (SE) will always exist. Having as objective the creation of a co-simulation in which each application executes into its native environment, the need for bridging separate simulations, the ability of unified control and coordination, as well of data sharing between them, becomes of great significance. To meet these demands, adaptation of non-SMP2 compliant space applications to the SE is indispensable. Creating a standardized interface and enabling this interconnection includes deploying an SMP2 specific interface and an application specific plugin to expedite the two parties' communication. SimBridge, is an SMP2 meta-model, designed to provide a flexible and seamless interfacing for SMP2 SEs. It is a configurable, scalable, and easy-to-use software that can interconnect any external application (EA), just by providing a language–specific library or toolbox to handle the message transfer for each such application. Currently, SimBridge binds the SIMSAT engine with external applications via a TCP connection and provides full-fledged services as: XML message interchange, parsing and evaluation, requests' handling, synchronization and execution, and finally returning results to the origin.

INTRODUCTION

System modeling and simulation tools are pivotal technologies in the engineering process for aerospace systems. As multitude of stakeholders are involved and different disciplines intersect, multiple views of engineering models are revealed. Furthermore, there are systems modeling requirements that demand high fidelity and necessitate the use of other environments or frameworks that are specialized for implementation, visualization and/or reporting purposes. For instance, in the Functional Engineering phase of an aerospace project, MATLAB/Simulink is often used by Model Supplier teams to deploy cutting-edge simulation models. For calibration and validation purposes, these models could be exploited across projects in several phases of System Engineering, as in Mission Performance Simulators (MPS) and Software Validation Facilities (SVF) with or without modifications and upgraded fidelity. Thus, the ability to integrate models from commercial off-the-shelf (COTS) tools or open source software (OSS) into a space simulator, such as SIMSAT, is becoming increasingly important.

Multiple efforts have been carried out for standardization in the field of space operational simulations. The most notable are the SMP2 standard and the REFA developed by ESA/ESOC in an attempt to standardize model interconnection and communication along with the general architecture of a generic spacecraft simulation. Furthermore, ESA/ESOC has also attempted to define a standard that allows formalized data storage and distribution of simulation assets [10]. In recent years, notable efforts outside ESA have also been made to define standard and meta-modeling paradigms in the field of

space operational simulations. Firstly, a study was undertaken in 2010 [9] that had to do with the development of a Python and C++ hybrid simulation based on the SMP2 standard. According to this simulation, a blueprint for defining interfaces between the simulation models and the framework has been described. Finally, an approach for model integration with the common simulation framework SIRIUS is investigated in [8].

On the other hand, many efforts have been made to extend the capabilities of space simulators. A hybrid simulation technology is described in [1] that extends flight simulations by providing a dynamic programming interface and enables more than one programming languages within a simulation. Furthermore, as most individual engineering disciplines have their own modeling methods and tools for design, analysis or simulation, there is need of an integrated model-based approach at the system level that ensures overall consistency and proper coordination. This concept is discussed in depth in [2]. Finally, the Dshell++ [3] is a physics-based simulation framework that wraps a Python interface around the C++ classes so that simulation setup and control can be completely script-driven. However, the space industry is not unique in these aspects. Similar challenges are present in other industrial sectors where complex systems are produced, such as defense, power systems' simulations, etc. More information for the aspect of co-simulation platforms for smart-grid applications can be found in [4]. The present paper focuses on the bridging concept of space simulators with various external tools and applications.

The challenge in the concept of bridging space simulators is to create a solution that would integrate a variety of simulators, modeling frameworks, databases, visualizers and reporting systems into a simulation that is distributed across various nodes. Another activity associated with this challenge is the identification of the EAs that offer an application programming interface (API) or a communication link and could benefit from a bridging with a space simulator, and vice versa. A third challenge related to the bridging concept is the plethora of EAs to take into account and the correspondingly large variety of communication protocols that are used. There are several ways to integrate simulations, including High Level Architecture (HLA) [11], which was developed specifically for distributed simulation systems, and standard inter-process communication techniques and protocols including, shared memory, TCP/IP or COM. Lastly, an important issue in the creation of a co-simulation environment is the synchronization of different simulators with separate clocks and timing engines. In that case, data exchange must take place in conjunction with the associated timestamp (e.g. simulation time), in order to specify the time for which data are valid.

The SimBridge infrastructure aims to interconnect External Applications (EAs) with Simulation Environments (SEs) and attempts to address all the aforementioned problematic areas. In Fig. 1 the SimBridge concept is depicted. Currently, development is mainly focused in the interconnection of several EAs such as Celestia and Python scripts with the SIMSAT SE. The SIMSAT environment is aimed to be supported both in the SMP2 space, i.e. simulation models, and the simulation's kernel to support the control of the simulation via CORBA mechanisms. For example, a model can be rapidly developed using Octave and immediately used within the SE itself with no conversion. Thus, the bridging concept is considered of high value in early stages of a simulator's design and development, since engineers of varying scientific fields can test their models within the SE with little to no effort and only minimal knowledge of the SMP2 standard and its inner workings. The bridging concept can also be used from the SMP2 models in order for them to use external functionalities. For example, an SMP2 model could use the bridge in order to execute an external Python script and get its results. This allows developers to make usage of higher level functionalities that are offered from external tools and operate outside the context of the SMP2 standard for computation purposes.

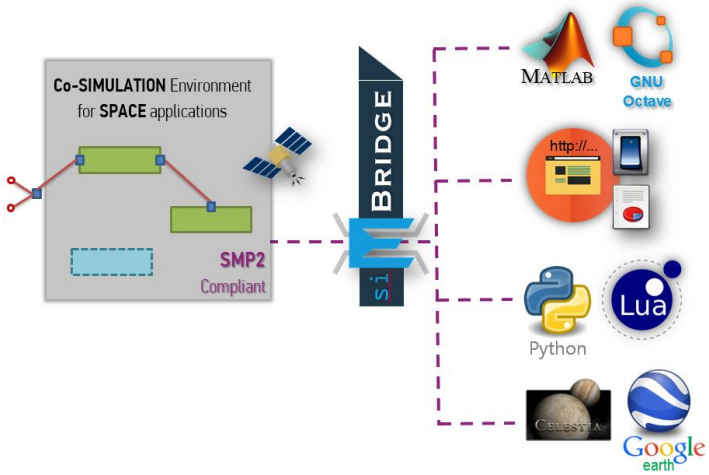

Fig. 1. SimBridge Concept

This paper is organized as follows; in section “SimBridge Application”, the major requirements of SimBridge application are described and a delineation of the SimBridge’s major elements is given. A demonstration of SimBridge usage in conjunction with Python scripts, and a Celestia application based on Lua scripts, is provided in section “Use Case”. Finally, in section “Future Steps”, our view for SimBridge’s coming expansions is cited.

SIMBRIDGE APPLICATION

In order for the SimBridge architecture to be imported in the SIMSAT infrastructure all the appropriate space standards are applied in terms of documentation and project workflow. More specifically, for aspects concerning the space software engineering including requirements definition, design, production, verification and validation the ECSS-E-ST-40C was followed.

Major Requirements

The principal functionality of the SimBridge software is to provide bidirectional interfacing between SEs supporting the SMP2 standard and non-compliant to space standards EAs. Currently, the SimBridge application is bound to SIMSAT and acts as an SMP2 meta-model. Bidirectional interfacing involves handling requests with direction from EAs to SIMSAT (incoming requests), as well as requests from SIMSAT to the EAs (outgoing requests). More specifically, the major requirements of the bridging concept are:

- Simulator’s data retrieval/modification. SMP2 published fields shall be able to be delivered/modified to/by the EAs and SMP2 services can be used to retrieve kernel data such as latency, speed factor, and simulation state.
- EA’s data retrieval/modification from the simulator.
- SMP2 method’s execution and events scheduling under EA’s demand.
- Execution of external functionality under simulator’s request.
- Formalized message’s structure that contains simulation timestamp for data produced in order to synchronize and coordinate the co-simulation.

The SimBridge software provides all aforementioned features but is not limited to them. An additional SimBridge feature is the ability to perform the data retrieval or the functionality execution in a synchronous or asynchronous way. A synchronous operation dictates that the incoming/outgoing request will be performed via direct function calls and the thread that initiated the operation will wait for the task to finish before continuing. On the other hand, an asynchronous operation implies a low priority request and the caller will not pause upon task’s completion. The goal is to free the main thread so that it can continue to respond to other requests, rather than freezing. Thus, the asynchronous mode of operation is a feature that can accelerate the system’s throughput, as slow or expensive processes shall run asynchronously on a background thread.

Moreover, the SimBridge application supports the widely used TCP/IP protocol and provides an application specific library to facilitate the EA’s connection. Although TCP/IP protocol is its primary communication protocol, the SimBridge is not limited to its usage. Instead it is designed in such abstract way, that it can be easily extended and support additional communication protocols, such as CORBA and HTTP or inter-process communication techniques, such as shared memory and pipes. Currently, the requests are exchanged through TCP sockets in plain text encoded in UTF-8 and formatted in XML. This serves three purposes; first, the data remain structured with XML and can be serialized and de-serialized with little effort; second, they can be transferred through sockets or any other inter-process communication mechanism in a platform-independent manner; and third, they can be easily inspected by a human operator during troubleshooting.

Trade Offs Towards Architecture Definition

During the requirements’ definition phase, a logical design of the infrastructure had to be achieved and several decisions had to be made in terms of the complexity and the functionality of the software. The first issue that had to be resolved was to determine where the software shall exist and how it shall connect with the SEs and the EAs. Initially, it was considered that there should be one persistent instance of the infrastructure’s “core” that resides on a server where the SEs connect to it via specific. The EAs were considered to utilize communication protocols such as TCP sockets, cloud

services etc. to connect to the infrastructure. However, this approach was considered to not be scalable enough by adding many obstacles and constraints and also required a server that could be used. It was instead decided to have one SimBridge instance per SE while allowing this instances to easily communicate among them with the use of the communication mechanisms.

The second design decision that had to be made was in regards with how the aforementioned SimBridge instance is created. Since the SIMSAT SE was the target platform of the first version of the framework, there were two options. The first one was to have a CORBA component that shall reside in the simulation's environment kernel and the second choice was to have an SMP2 model or class that shall reside in the simulator itself. While both options are viable, it was decided that an SMP2 model should be primarily used. There are several reasons that led to this decision. Firstly, it was considered that the users of the EAs will primarily desire to exchange data and control the simulator's components. While that is also possible from a SIMSAT kernel component, it was deemed more straightforward from the perspective of an SMP2 model as it has direct access to all the desired interfaces and does not rely on direct CORBA calls as would be the case from the perspective of a SIMSAT component. Furthermore, the SMP2 adapter provides all the main functionalities of the kernel to the models and thus enables the models to have information and some control over timing information, speed factor, simulation state etc. Last but not least, the SMP2 model was considered more "generic" in terms of implementation. As many SEs adopt the SMP2 standard, the model can be easily ported to other environments with minor modifications.

Another issue under consideration was the system's configuration. Generally many file formats are used, such as CSV, XML, binary or YAML. The XML format was finally chosen as it can be strictly defined and easier to be validated due to XML schemas. Although XML files are usually large in extend, it is easier to be processed and more human readable as they are string based files.

Architectural Design

The SimBridge infrastructure is divided in four core modules, namely, the Interface Module that is responsible for the connection with the SE, the Communication Module that is responsible for the connection of EAs with the infrastructure, the CLI Module that is responsible for auxiliary facilities such as logging and user control of the infrastructure and the Management Module that is responsible for the "core" functionalities such as marshalling, scheduling, parsing etc. An overview of the SimBridge logical architecture is presented in Fig. 2.

The Interface Module is the communication channel between the SE and the rest SimBridge components. It supports interfacing of SMP2 space where SMP2 models and SMP2 adapter exist, and kernel space interfacing, which for the SIMSAT case CORBA technology is applied. Its first responsibility is to provide simulator's models data or operations requested from the EAs via the Dynamic Invocation mechanism. Its second responsibility is to provide an appropriate interface to the simulator's models so that they can request data or operations from EAs.

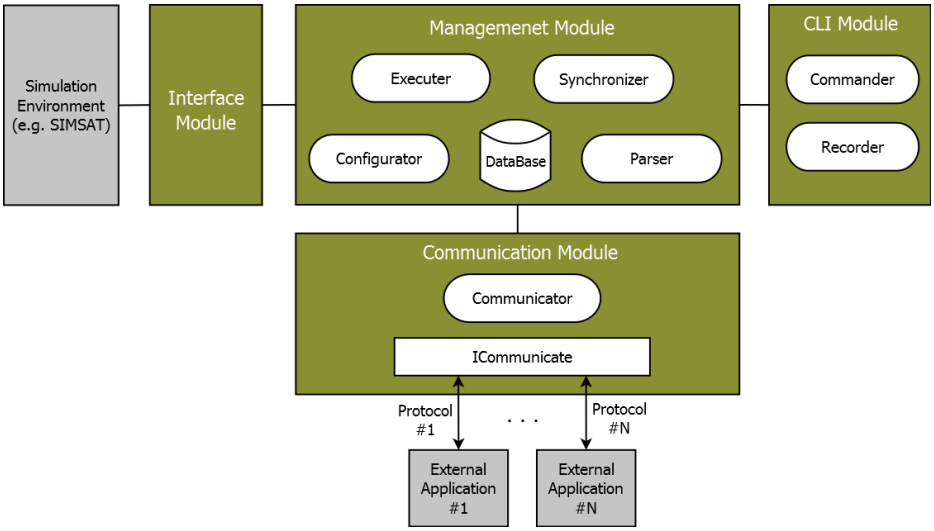

Fig. 2. SimBridge architecture

The Management Module is the main component of the SimBridge infrastructure and acts as its kernel. Its first responsibility is to synchronize and execute the requests that are made either from the SE or the EAs. This is achieved by the Synchronizer and the Executer components. The Synchronizer component holds all the requests that are to be executed in a list and according to a time-base it removes requests from this list and provides them to the Executer for handling. The Synchronizer can be configured by the user in order to provide a requests' lifespan and define priorities according to a request's type and lifespan. The Executer component is responsible for executing the requests provided by the Synchronizer. To that end, it utilizes the Interface Module for requests that are targeted to the SE or the Communication Module in the case that the request is targeted to an EA. Since requests are generally independent from each other and they can be of synchronous nature, thus locking the Executer until their execution is completed, the Executer features a user defined number of threads in order to handle them in a fast and parallel manner.

Since communication among the SEs and the EAs is based on XML files, the Management Module offers an XML Parser and a Database component. The Parser component is responsible for reading and writing these XML files based on strict XML schemas. The Parser also provides a set of helper functions to the rest of the infrastructure that allow the retrieval of certain information structures. Finally, the Parser makes heavy use of the Database component in order to store the required information. Currently, the Database component is a set of maps and vectors that are used to store the configuration file of the SimBridge infrastructure and the configuration files of the EAs. The Configurator component processes configuration files and performs respective initializations of the SimBridge application.

The Communication Module is responsible for the communication of the SimBridge infrastructure with the EAs. It is comprised by one main component and an interface. The main component of this module is the Communicator and is responsible for instantiating all the available communication protocols, providing incoming requests to the Synchronizer for execution and dispatching outgoing requests to the appropriate EA. The ICommunicate interface provides a set of functionalities that shall be provided by every available communication protocol and as a result each communication protocol shall implement it. Through the last Module named CLI, a human operator has the ability to manage external connections, to control SimBridge's operation and invoke external model commands during runtime. All this issues are addressed from the Commander sub-element of CLI Module. The second sub-element named Recorder, is responsible for printing informational, debug and error messages regarding the particular SimBridge instance.

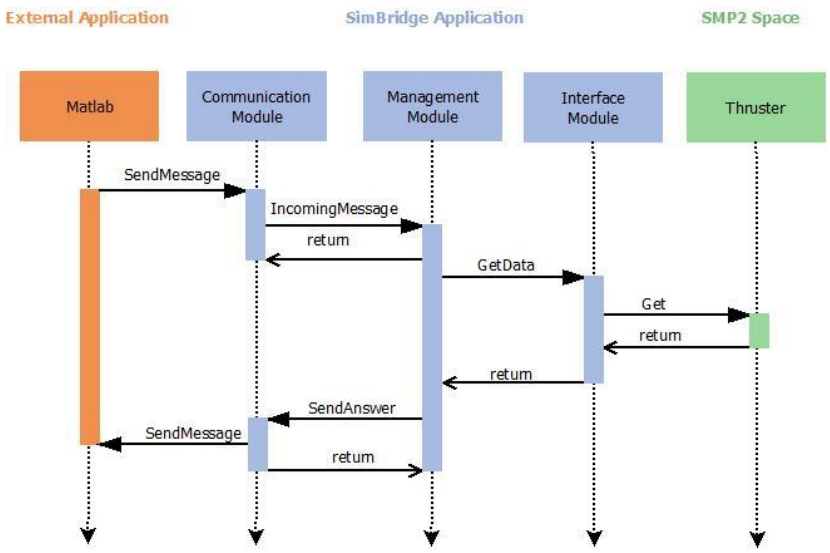

Fig. 3. Sequence diagram of incoming request to be handled in a synchronous way

Fig. 3 depicts a simplistic paradigm of the actions' sequence that have to take place in order to an EA achieve to obtain data coming from an SMP2 model in a synchronous way and via the SimBridge application. Since the request is about to be executed in a synchronous way, the EA shall pause its execution and wait until the answer's arrival. The request can be matched only with an answer due to the message's unique identifier that it carries. In SimBridge's side, after the Communication Module delivers the new incoming request to the Management Module, it returns back to listen for more connections and new messages. The Executer's thread that is about to handle a synchronous request shall ask the data

from the Interface Module, and it will remain idle for as long as it takes to the Interface Module to return the data. As soon as the data are delivered to the Management Module, it shall create a new answer message and forward it to the EA through the Communication Module.

USE CASE

In order to demonstrate the capabilities of this novel tool, a simple demonstration configuration was assembled. This is aimed to demonstrate three separate capabilities: First, to implement a model's functionality outside the SMP2 space, secondly to control said model from an EA, and third, to draw data in order to use them for visualization. The system was composed of the following components executed in parallel:

- The SIMSAT program, including a SimBridge instance and an internal SMP2 model.
- A Python script that implements the above model's functionality.
- The Celestia application to visualize the position and direction of the model.

The Python script as well as the Celestia application connect to the simulation via the SimBridge model using sockets. Python was chosen as a non-compiled scripting language in order to demonstrate its use for quickly prototyping model functionality without re-compiling the whole simulation. For this test, the Python script implements a simplistic calculation for the model's rotation movement. To that end, a special Python toolbox was developed that was responsible for connecting to the SimBridge, monitoring the connection for incoming requests and parsing and constructing messages as required by the application. Finally, inside the script, a command line interface was also implemented in order to affect the rotation parameters externally at runtime. In Fig. 4 a sample of the XML messages that Python sends to SimBridge is given. In this XML message the Python application requests the modification of an SMP2 Field named "m_position" that belongs to the Position Model to the value of three.

```
<?xml version="1.0" encoding="UTF8"?>
<Message>
    <ID>23</ID>
    <Sender>Python</Sender>
    <Recipient>SimBridge.0</Recipient>
    <MessageType>REQUEST</MessageType>
    <ErrorCode>0</ErrorCode>
    <SimulationTimestamp>1000000</SimulationTimestamp>
    <Request>
        <RequestType>SET</RequestType>
        <SchedulingInfo>
            <How>SYNCHRONOUS</How>
            <ClockType/>
            <Period/>
            <StartOffset/>
            <Reps/>
        </SchedulingInfo>
        <Transactions count="1">
            <Transaction>
                <TransactionOwner>PositionModel</TransactionOwner>
                <TransactionName>m_position</TransactionName>
                <TransactionReturnType>VOID</TransactionReturnType>
                <Parameters count="1">
                    <Parameter>
                        <ParameterType>INT16</ParameterType>
                        <ParameterValue>3</ParameterValue>
                    </Parameter>
                </Parameters>
            </Transaction>
        </Transactions>
    </Request>
</Message>
```

Fig. 4. XML message to set value in an SMP2 field

Celestia is a freeware and cross-platform space SE that allows users to navigate through space and view characteristics of planets, stars, comets and more. A real-time 3D visualization system for space simulations is also discussed in [5]. Celestia allows the creation of add-ons in order to display additional objects such as spacecraft or newly discovered stars. It is packed with a Lua interpreter that allows the addition of logic and control in user created add-ons.

Based on the latter, an add-on has been developed that displays a simple spacecraft orbiting earth. The add-on employs Lua's capabilities to connect to the SimBridge infrastructure. When connected, it requests positional values from the appropriate SMP2 model and based on them it redraws the spacecraft accordingly. Fig. 5 depicts a small spacecraft in Celestia while orbiting earth along with the SIMSAT and a Python shell that control the spacecraft's rotation.

Through this use case, we firstly delegate an SMP2 model's functionality to an external Python script, while rendering in 3D the orbit and the axis rotation of the simulated spacecraft in Celestia. Since all communication is based to XML messages, implementation of such add-ons for EAs is user-friendly and easy to achieve as most, if not all, scripting and programming languages offer libraries to easily handle XML files. From user perspective, XML files are considered a common format to describe data structures and as a result their understanding and editing is considered trivial.

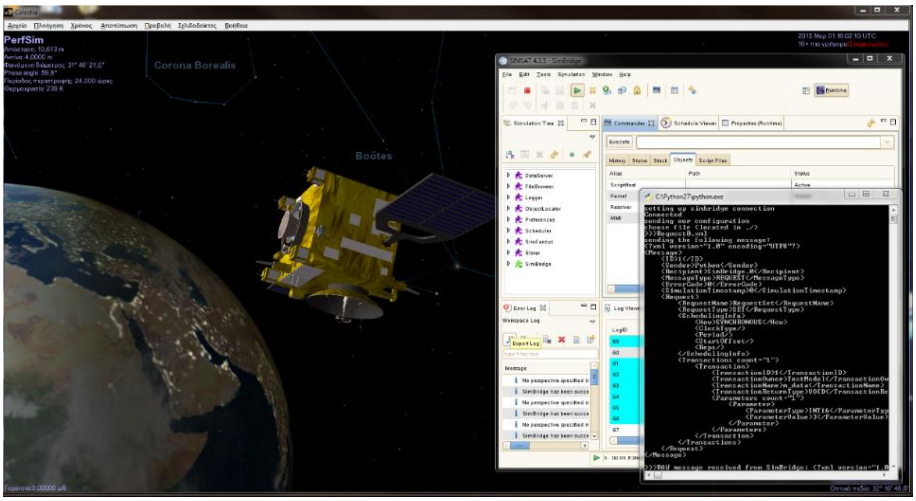

Fig. 5. Spacecraft in Celestia simulation environment

FUTURE STEPS

Even though the SimBridge infrastructure offers a plethora of features that ease the bridging concept between SEs and EAs, several areas of improvements and advancements are under consideration. First of all, the need to make the infrastructure more generic has been identified. To that end, the creation of appropriate plugins to support more SEs is considered. Specifically, support for the SimTG and the EUROSIM environment is currently investigated. Furthermore, the addition of new communication protocols and EAs is being evaluated. More communication protocols shall provide easier ways to EAs to connect to the infrastructure while more EAs shall provide more flexibility to end users.

Considerations for external tools are currently focused on reporting applications. Since SEs generally provide limited reporting facilities, it is made possible for the SimBridge to thrive in this context. By offering appropriate interfaces to support specialized visualization and reporting frameworks, it is made possible to end-users to have a more intuitive, user-friendly manner of controlling and monitoring the simulation. Several external tools are currently being considered to have fully featured connectivity with the SimBridge concept. Firstly, Celestia can offer spacecraft visualization and control. Furthermore, Microsoft Excel, or similar spreadsheet applications, can offer spreadsheet analysis of values as well as the creation of appropriate graphs and mathematical facilities to investigate, correlate and interpret simulation results. Finally, live reporting and monitoring can also be made possible via web services and custom made web-applications. Web infrastructure can also be used to control the simulation allowing users to both monitor and operate the SE.

An issue that is also currently under investigation is the performance of the SimBridge infrastructure. It is considered that since the SimBridge acts as an extra layer between EAs and SEs it has an impact on the overall performance of the simulation. To that end, SimBridge shall be enhanced to provide a solid integration with the C-PDES engine [6,7]. Specifically, the C-PDES engine's Communication Manager could be utilized to provide synchronized and optimized communication among instances of the simulation's kernel such as SIMSAT's Master/Site configuration. Furthermore, in order to achieve higher parallelization and performance the SimBridge plugin that refers to the SE should offer multiple SMP2 events that can then be exploited by the PDES engine. Lastly, advanced features of the PDES engine such as

causality error detection and synchronization policies can be utilized by SimBridge to resolve internal timing constraints and timing issues. More specifically, since the PDES engine allows the creation of custom synchronization policies, it is considered to implement and integrate policies that are related to prioritization and lifespan of events in order to better support the underlying requirements of the SimBridge infrastructure.

Finally, the SimBridge infrastructure could be used to interconnect multiple SEs. Using the bridging technology, it could be possible, for example, to use a SIMSAT based simulator while parts of it operate in a SimTG environment with Hardware in the Loop. This however, is a complex concept which, even though it is supported in theory, it is difficult to operate. First of all, there can be timing issues that can cause causality errors and severely damage the simulation's correctness and fidelity. As previously discussed, however, solutions to avoid such condition already exist and can be utilized via the C-PDES engine.

CONCLUSION

The construction of a consolidated co-simulation where the simulators will coexist with external tools and applications is widely spread in the aerospace community. The SimBridge application attempts to address the challenges in this domain. Through this innovative bridging the simulation and the EA work together and the latter can obtain or modify simulation variables' state, send commands and schedule events for execution, or even exploit all the simulation services. Furthermore, the simulation models have the same benefits by interfaced with SimBridge, as its services are bidirectional and can perform in a synchronous or asynchronous mode. Multiple external connections are supported at simulation run-time and multithreading techniques are exploited to speed up the execution, raise requests' throughput, and eliminate idle time. Due to SimBridge technology, a great variety of EAs such as MATLAB, Celestia, Network Simulator 2/3, Web Services, and Python applications collaborate with SIMSAT to easily create a co-simulation environment. Hence, simulation models' rapid prototyping and validation are enhanced, the overall human interaction with the simulation is significantly upgraded and interdisciplinarity in the European space community is augmented.

REFERENCES

[1] Scott M. Nemeth, "Hybrid Simulation Technology: The Next Step in the Evolution of Spaceflight Simulations", *SpaceOps Conference*, 2008.

[2] Harald Eisenmann, Juan Miro, Hans Peter de Koning, "MBSE for European Space-Systems Development", *INCOSE*, vol. 12, Issue 4, pp. 47-53, December 2009.

[3] Christopher S.Lim, Abhinandan Jain, "Dshell++: A Component Based, Reusable Space System Simulation Framework", *IEEE SMC-IT Conference*, July 2009.

[4] Hua Lin, Santhoshkumar Sambamoorthy, Sandeep Shukla, James Thorp, Lamine Mini, "Power System and Communication Network Co-Simulation for Smart Grid Applications", *IEEE ISGT Conference,* January 2011.

[5] Marc I.Pomerantz, Abhinandan Jain, Steven Myint, "Dspace: Real-time 3D Visualization System for Spacecraft Dynamics Simulation".

[6] Vemund Reggestad, Nikolaos-A.I.Livanos, Pantelis Antoniou, Ioannis E.Venetis, "Operational Simulator Going Parallel: From a Dream to a Concrete Concept", DASIA Conference 2011.

[7] Nikolaos-A.I.Livanos, I.Angelis, V.Alifragkis, S.Hammal, A.Walsh, V.Reggestad, M.Pantoquilho, "Performance Optimization Framework for Spacecraft Operational Simulators", *European Ground System Architecture Workshop*, June 2013.

[8] Ulrich Sennes, Roland Schabenberger, "Model Integration with the Common Simulation Framework SIRIUS", *AIAA Modeling and Simulation Technologies Conference,* August 2011.

[9] Scott M.Nemeth, Peter Demarest, "Research and Development in Application of the Simulation Model Portability Standard", *SpaceOps Conference,* 2010.

[10] ESA/ESTEC, ECSS-E-TM-10-23A, "Space engineering, Space system data repository", November 2011-23A.

[11] C.A.Boer, A.Bruin, A.Verbraeck, "A survey on distributed simulation in industry", *Journal of Simulation,* vol.3, pp.3-16, 2009.