An underwater buoyancy-driven glider simulator with Modelling & Simulation as a Service architecture

Manlio Oddone[1*], Agostino Bruzzone[2], Emanuel Coelho[1], Daniele Cecchi[1], Bartolome Garau[1],

[1]NATO STO Centre for Maritime Research and Experimentation, Viale S. Bartolomeo 400, 19126 La Spezia, Italy,
[2]DIME, Univ. of Genoa, Genoa, Italy

*E-mail: manlio.oddone@cmre.nato.int.

Abstract

This paper describes the architecture of an underwater glider simulator applying the Modelling & Simulation as a Service (MSaaS) paradigm. The simulator implements a modular and scalable service-oriented architecture where Web services are employed for the underwater glider kinematic and hydrodynamic models, for the Oceanographic models forecast data, for the glider motion control system and for the motion behaviours. The simulator itself is offered as a Web service, using a browser based GUI to present the status and result of the simulation. Two main problems are targeted: 1) the estimation of the underwater trajectory of the glider between two known (measured) surfacing points and 2) the prediction of the glider future surface position given its past history, its motion behaviour and the Oceanographic Model forecast. Considerations about the estimation of the motion of the glider when the corresponding motion control and behaviours algorithms are not disclosed are done, capitalising on the work described in a previous paper and an alternative approach to simulating the unknown behaviours, considering the hardware-in-the-loop, is discussed. The discussion also involves architectural, and technological aspects such inter-process and cross-language communication and hardware-in-the-loop interfacing.

Keywords: Underwater glider, MSaaS, Web services, AUV, Modelling & Simulation, M&S, REST

1 Introduction

This paper describes the architecture of an underwater glider simulator implementing the Modelling & Simulation as a Service (MSaaS) paradigm with Command and Control (C2) and High Level Architecture (HLA) interoperability.
The simulator is provided as a Web service. The Graphical User Interface (GUI) is decoupled from the simulator engine and is provided either in the form of a Web GUI: a Javascript application with mapping display and graphical capabilities, or as an HLA Virtual-Reality graphical console. The glider dynamic and kinematic models, the models of the navigation control system (*front-seat*), the mission behaviours (*back-seat*) and the environment models are implemented as Web services.
The simulator output is provided using open standards for maximum interoperability, with NATO simulators, C2 systems, navigation/mapping/GIS software, Web applications and scientific data analysis/processing tools.
The simulator engine implements a three-phase discrete event processing core with a hybrid continuous/discrete model approach that makes the simulation suitable for constructive as well as virtual simulation applications. It can be configured for a range of applications, including:

- **Decision support tools** for optimisation and risk mitigation during mission planning and execution
- **Post mission analysis and debriefing tool** to analyse, reproduce and explain what happened during a mission in the post mission analysis phase
- **Test bed** for development and tuning of models, path planning and optimisation algorithms
- **Virtual reality applications** for personnel training and mission preparation
- **Hardware in The Loop Virtual Simulation** for mission behaviours safe testing on the bench before real deployment in the field

Despite the apparent complexity, the architecture is an ensemble of simple subsystems that are easy to manage and to secure. The implementation as a Web service makes the tool easily deployable and maintainable. Once the system is deployed on a server (e.g. as a virtual machine or as a container-based virtualisation solution) user may access it from workstations and mobile device with a Web browser. No installation or configuration of the tool is required by the user.
Flexibility and scalability makes the simulator engine suitable for cloud deployment, remotely accessed through a Wide Area Network (WAN) such as the Inter-

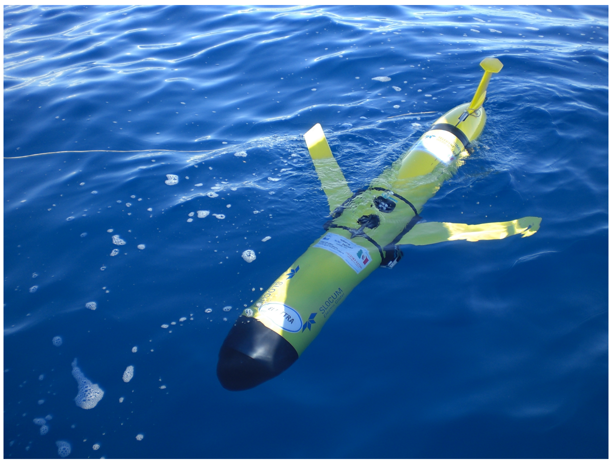

Figure 1: An underwater glider

net, or deployed on a local server and accessed by users on from a Local Area Network (LAN).

1.1 What is an underwater glider

Underwater gliders (see figure 1) are a special class of Autonomous Underwater Vehicle (AUV) that use small changes in their buoyancy to achieve vertical movement. Wings and control surfaces convert the vertical velocity into forward velocity so that the vehicle glides downward when denser than water and glides upward when buoyant. Due to the buoyancy-driven propulsion their motion in the vertical plane follows a saw-tooth profile as depicted in figure 2.

Glider motion is extremely high energy efficient, for this reason, despite the slow speed, they achieve particularly long-range capabilities making them ideally suited for scientific data collection subsurface missions, sampling environmental parameters at regular intervals at a regional or even larger scale.

They can be programmed to autonomously navigate for weeks or months at a time. When they surface they transmit the acquired data to shore, using satellite communication. At the same time they download new missions with substantial cost savings compared to traditional ship-based research methods.

Glider motion is influenced by currents and changes in the water density. Before starting a mission a glider must be carefully ballasted to match the water density of the mission scenario area. An improper calibration of the buoyancy could compromise the glider capability to perform its mission and in the worst cases could even lead to the loss of the glider.

1.2 Why a glider simulator

Underwater gliders operate in ocean environments characterised by a complex spatial variability. Due to their sensitivity to environmental parameter change, monitoring of the glider state is essential to achieve mission success by minimising risks due to adverse environments [19]. By predicting environmental impact on glider navigation, the pilot may decide to modify the mission plan

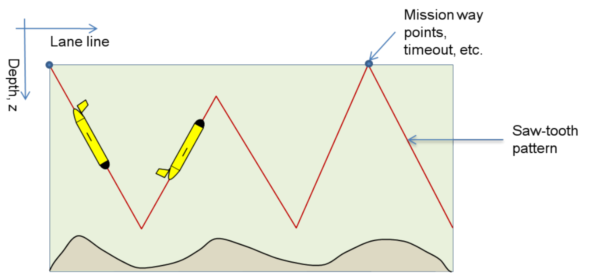

Figure 2: Saw-tooth motion pattern of an underwater glider

in advance to operate with the minimum risk of asset loss [13].

A simulator capable to accurately reproduce a glider behaviour within a scenario with a time-variable environment is considered and in particular two main problems are targeted:

- The estimation of the underwater trajectory of the glider between two known (measured) surfacing points
- The prediction of the glider future surface position given its past history, its motion behaviour and the Oceanographic Model forecast

This work aims to improve a Decision Support Framework[13] implemented at the NATO Science and Technology Organization - Centre for Maritime Research and Experimentation (STO-CMRE) by providing a simulator tool that would allow extending the developed glider model to the 3D case, taking into account the vertical changes of water density, model the glider at the surface, taking into account the effect of waves and winds and finally implement the glider guidance system to compensate for current prediction integration.

The simulator would also serve as a testing Framework for Path Planning algorithms of AUVs in realistic ocean environmental fields [15] and as a Virtual-Reality and environmental simulator with real Hardware in the Loop.

1.3 HLA interoperability

High Level Architecture (HLA)[14] is a general purpose architecture for the interoperability between distributed simulation systems. It is the subject of the NATO standardization agreement (STANAG 4603)[8] for modelling and simulation. In an HLA architecture multiple federates are composed into a federation exchanging object attributes and interactions through a Run-Time Infrastructure (RTI). An RTI is basically a bus connecting the federates with a publish-subscribe model.

Federates joining a federation may subscribe to an object class and then discover objects of such class and receive attribute value updates. Interactions are similar to objects except that an object is persistent (e.g. a tanker) while an interaction is only used once (e.g. an ammunition detonation, or a collision between vehicles).

HLA requires a shared reference data exchange model

used as an agreement about the data (objects and interactions along with their attributes and parameters) that will be exchanged within the federation. Such model is called Federation Object Model (FOM). The modular RPR FOM (Real-time Platform Reference FOM) and the more recent NETN FOM (NATO Education Training Network FOM) provide building blocks for creating federation agreements.
HLA interoperability has been achieved by simulators developed at CMRE within the MCWS-MSTPA federation [5].
Interoperability between High Level Architecture (HLA) with Robot Operating System (ROS) [17] based autonomous systems has been achieved in CMRE in the work described in [3] where an HLA-based link between simulation and a SPARUS II AUV has been implemented and the integration into an HLA federation has been tested in a port protection scenario. For such scenario the hardware and software of the AUV has been included in the federation together with a *virtual simulator*. This works aims to extend such architecture by integrating the service-oriented concept.

1.4 Service-Oriented Architecture

The simulator architecture follows MSaaS paradigm that is a means of delivering Modelling & Simulation (M&S) applications, capabilities and associated data on demand by providers to consumers.
It has been designed according with the "MSaaS as a SOA" perspective described in the study on *Service-Oriented Architectures for Modelling and Simulation*[6] provided by the NATO Modelling and Simulation Group (NMSG) Specialist Team MSG-131.
The "MSaaS as a SOA" perspective looks to use Service-Oriented Architecture as the architectural approach for connecting and combining M&S service whose central aspects are:

- Communication following standards
- Loose coupling
- Interoperability and composability of services (to form new and maybe more complex services)

The implementation of the architecture has been done using Web services. The simulation engine itself is implemented as Web service.
Web services are server applications, that can be queried on a network (Internet or private) by client applications in order to provide a service, in the form of an exchange of information, and are not tied to any one operating system or programming language.

1.4.1 Types of Web Services

There are two main implementations of Web services: SOAP based and REST Web services.
Web services based on the Simple Object Access Protocol (SOAP) are commonly known as Web Services (WS or WS-*). WS exchange structured information using an eXented Markup Language (XML) Information Set for its message format, and relies on application layer protocols, most often HyperText Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.
WS technologies have been successfully used for simplifying interoperability while providing scalability and flexibility in multiple applications, including distributed simulation software[16] however they have been superseded by the more modern and lightweight REpresentational Stat Transfer (REST) or RESTful Web services[23].
REST is not a protocol but an architectural style that allows to manipulating textual representations of Web resources using a uniform and predefined set of stateless operations. A "Web resource" is any entity that can be identified, named, addressed or handled, in any way whatsoever, on the Web, identified by its Unique Resource Identifier (URI).
REST employs the uniform and predefined set of HTTP operations (GET, POST, PUT, DELETE) to allow a Web service to access and manipulate Web resources through their textual representation: requests made to a resource URI will produce a response in XML, HTL or JSON. Use of such well known and widely supported text formats, transmitted over the most widely used internet HTTP protocol is well digested by the most complex network configurations employing firewalls and proxies and allows for the maximum interoperability between systems over the internet.
REST operations are stateless, each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client and has to be passed to the Web service when it's functionality is invoked.
The main advantages of REST over SOAP are:

- REST services are easier to write and use since they implement the architectural style of the Web itself
- a RESTful Web service response may be in XML, HTML, JSON or any other defined format while SOAP only allows XML
- REST doesn't have the overhead of headers and additional layers of SOAP elements on the XML payload and thus requires less bandwidth and is faster to decode

The disadvantages of REST with respect to SOAP are:

- SOAP can use almost any transport protocol while REST uses only HTTP/HTTPs
- REST uses transport level security inheriting the security measures from the transport layer, such as Transport Layer Security (TLS) or Secure Socket Layer (SSL) encryption protocols using HTTPs, while SOAP, in addition, can use its own security mechanism, WS-Security, that offers protection from the creation of the message to its consumption
- SOAP has comprehensive support for ACID (Atomicity, Consistency, Isolation, e Durability) for short-

lived transactions and Compensation Based Transaction Management for long-running transactions and also supports two-phases commit across distributed resources while REST doesn't
- SOAP support WS-ReliableMessaging for a guaranteed level of reliability of message exchanging while REST expects clients to deal with communication failures by retrying
- SOAP has a formal mechanism to describe the service using the Web Service Description Language (WSDL) thah allows to discover the service and generate a usable client proxy automatically

RESTful design is therefore appropriate when the following conditions are met:

- The Web services are completely stateless. A good test is to consider whether the interaction can survive a restart of the server
- The service producer and service consumer have a mutual understanding of the context and content being passed along
- Bandwidth needs to be limited
- A caching infrastructure can be leveraged for performance
- Easy of service delivery or aggregation is a requirement

1.5 Organisation of the paper

In Chapter 2 we will present an overview of the glider model requirements including considerations for Hardware in the Loop simulation, helping to explain the target architecture presented in Chapter 3; in Chapter 4 we will better detail the architecture of a Web service; in Chapter 5 we will see an extension of the architecture from the point of view of security; finally in Chapter 6 we will give an overview to the Graphical User Interface;

2 Glider model requirements

A glider simulator my come to hand in several applications, such as:

- **Decision support tools** where the simulator is used during planning and execution phases of a mission to explore the possible outcomes in a safe simulated environment, in order to optimize the mission goals while minimising the risk of asset loss or mission failure
- **Post mission analysis and debriefing tool** to reproduce and analyse what happened during a mission and to explore what would have happened if different decisions or conditions were applied
- **Test bed** for development and tuning of models, where the simulator is used to reproduce with fidelity the behaviour of a real unit in order to test path planning, motion control and optimisation algorithms under development
- **Virtual reality applications** where the simulator is used to reproduce the behaviour of a real glider introducing at the occurrence particular events, such as hardware/software failures or environmental anomalies, in order to train the personnel for a particular mission or to respond to particular events
- **Hardware in The Loop Virtual Simulation** to safely test missions and behaviours of a real glider hardware and software before deployment in the field

By an analysis of the use-cases provided, it is clear that the simulator has to be able to work either in fast-time mode and in real-time mode. Furthermore it should be able to work either as a *virtual simulator* with Man in The Loop or as a *constructive simulator*. The problems that it should be able to respond to are then ascribable to two main general cases:

- The prediction of glider future surface positions given its past history, its motion behaviour and the Oceanographic Model forecast
- The estimation of the underwater trajectory of the glider between two known (measured) surfacing points

Montecarlo simulation is required to account for the stochastic process in presence of noise or disturbance on the models inputs (e.g. assuming a variability on the environmental inputs due to the estimation errors of the environmental models or to account for the simulated sensors measurements noise).
For the estimation of the motion between two know points, forward and reverse techniques of smoothing could be used, in particular is interesting the application of the Unscented Rauch-Tung-Striebel Smoother[18] in which a separate backward smoothing pass is used for computing suitable corrections to the forward filtering result in order to obtain the smoothing solution.

2.0.1 Glider model

There is not a unique model that fits all the possible use-cases, for example for the accurate motion estimation in the three dimensional ocean environment within a virtual simulator, a 6 degree of freedom non linear hydrodynamic and kinematic model would be suitable to offer the required accuracy, while for a Decision Support tools, a faster, simplified stochastic model as used by [13] may offer better performances.
Depending on the use case the most appropriate model should be used.
Several 3D motion models are available in the literature [2] [11] [10] [22]. These models carefully describe how the glider moves as a result of the forces acting on it.
However to describe accurately the motion of the glider is also necessary to take into account the behaviour of the glider navigation control loop, usually called the *front-seat* and the higher level scripted mission behaviours which, all together, form the mission and are called the *back-seat*.
For the maximum accuracy, it's necessary to include models that, with the maximum degree of fidelity, repro-

duce the front-seat control loop and the back-seat algorithms.
The front-seat acts as a closed loop control that reads the navigation sensors on board the glider and generates control outputs toward the actuators of the rudder, ballast and bladder in order to obtain the desired motion of the glider. Gliders Sensor should also be modelled along with their measurement errors.
On top of the front-seat, an additional control layer, the back-seat is responsible to impose the glider a set of pre-programmed behaviours that together form the glider mission.
For back-seat modelling is also necessary to simulate glider communications and GPS fix readings.
For example communications would only be available after surfacing and GPS fix would only be taken after a variable acquisition time is elapsed during surfacing, with time to fix and communication ranges being affected by the weather conditions.
When the glider surfaces, a different dynamic/kinematic describes the gliders drifting as subject to wind, waves and surface currents.
Therefore a rather complete glider model is represented in figure 3 including the following sub-models:

- Glider Hydrodynamic Model: describe the forces acting on the glider while diving and climbing as a result of its buoyancy.
- Glider Surface Model: describe the motion of the glider while surfacing as a function of wind, waves and currents.
- Glider Kinematics Model: describe the motion of the glider.
- Glider Front Seat Model: describe the navigation control algorithms of the glider
- Glider Back Seat Model: describe high level mission behaviours of the glider: e.g. how it compensate a current.
- Simulated Sensor Outputs: provides the glider sensor outputs. E.g. pressure level, compass reading, Inertial Measurement Unit Output, GPS and the noise associated to the readings.
- Communications: long-range/low-bandwidth satellite communication and short-range/high-bandwidth HF radio communication.

2.1 Hardware in the Loop requirements

Unmanned Vehicles integrate a large number of complex hardware and software subsystem to manage the navigation and the execution of their missions. This level of complexity may be difficult if not impossible to describe with simplified models. Hardware in the Loop simulation comes to hand to test the real vehicle into a virtual environment before deployment in the real environment.
Significant work has been done at CMRE for the integration of HiL within the PARC project[3] for a SPARUS II AUV where the hardware and software of the AUV has been included in an HLA federation together with a virtual simulator. SPARUS II AUV is not a glider but a propeller powered underwater autonomous vehicle, it has a limited time/range autonomy and is less sensible to environmental conditions. Its software is implemented on top of the Robotic Operatin System (ROS)[17]. The approach used in this work was to bridge the ROS bus with the HLA federation by a ROS node which converts the information from ROS to HLA and vice versa.
Our approach to HiL capitalize on the work previously done within the PARC project, extending the concept by introducing a Web service that connecting to the glider hardware, feeds the sensors inputs (e.g. pressure, compass, GPS, IMU...) with simulated data, and uses actuators outputs to calculate the forces acting on the AUV to reproduce its motion in the virtual environment. Changes in water density, sea state, currents, wind and waves intensity and direction are considered by the dynamic and kinematic models to reproduce in the most accurate way the motion of the glider in the simulated environment. How well the HiL simulation reproduce the motion of the glider depends on the accuracy of the environmental and on the motion models of the glider.
HiL simulation could be used to test the correct functionality of the real glider subsystems, including its hardware components; to verify the correct implementation of the behaviours; to understand if a planned mission is feasible or not; and finally to train the personnel to the use of the systems in the most possible realistic conditions.
HiL simulation contributes to speed up development/test/validation time, reduce test/development costs and greatly minimize the risk during real missions execution.
HiL is also of great help when the algorithms of the behaviours and the navigation control loop are not known in the detail (for example for commercial AUV whose software is undisclosed) and have to be reproduced. In such cases, differences between the real and the reproduced behaviours and the control loop could lead to simulation results differ significantly from the real result.
A drawback of HiL simulations is that they are very slow. In most cases, unless the AUV implements a specific system to enable time acceleration, simulations with HiL requires to run in Real-Time. In the case of underwater glider missions, these could last for weeks or even months and the Real-Time execution may be inappropriate. In such cases HiL could still be used to explore a range of particularly significant cases that could be used to tune and then validate the parameters of the equivalent models which could then reproduce the AUV behaviours in fast-time simulations.

3 Simulation Engine architecture

The work done on the definition of the target architecture follows the recommendations of MSG-131[6] on development of MSaaS target architectures. The resulting target architecture is derived from the SD VIntEL NATO Reference Architecture and has the following characteristics:

- Interoperable

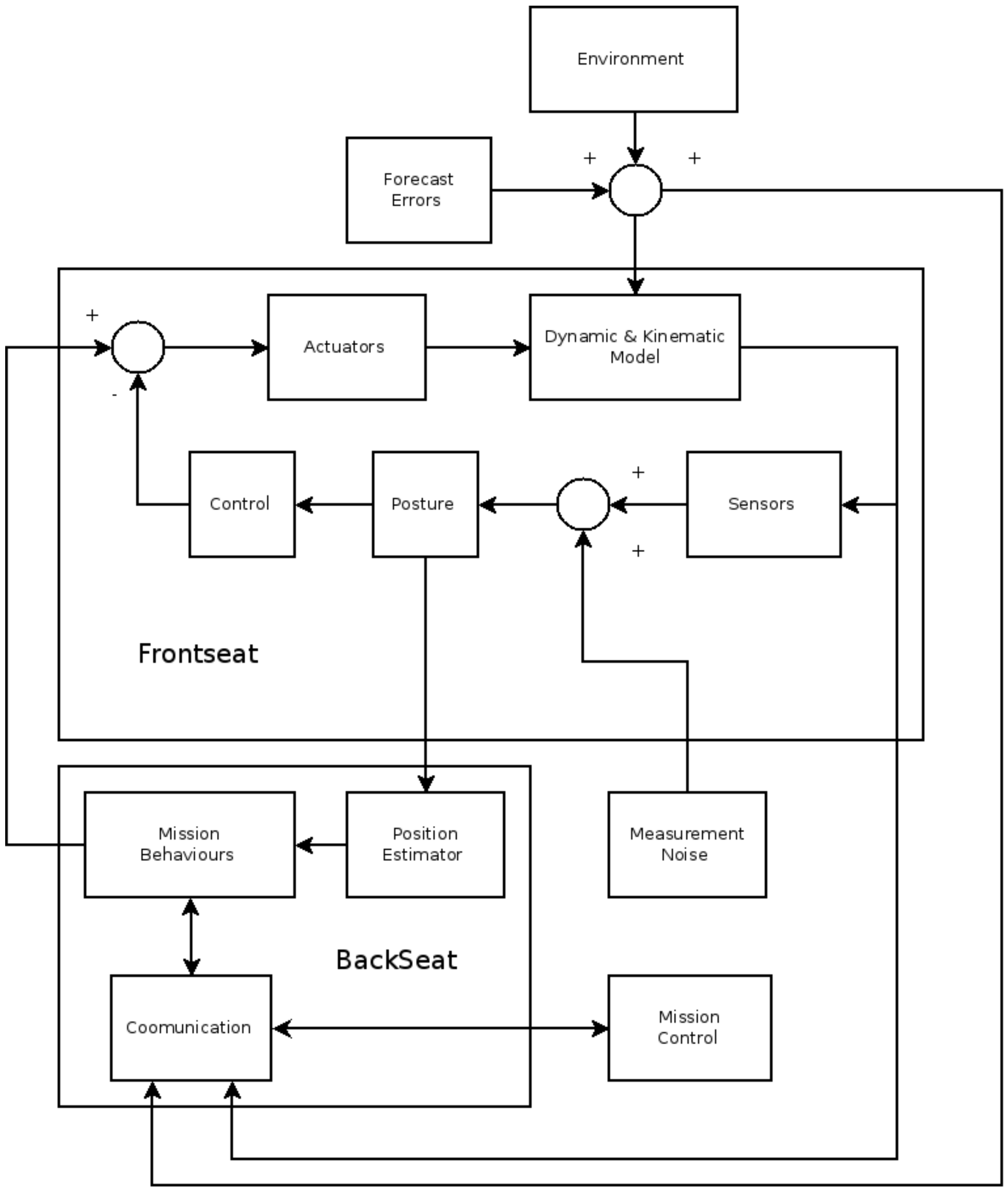

Figure 3: Glider model block diagram

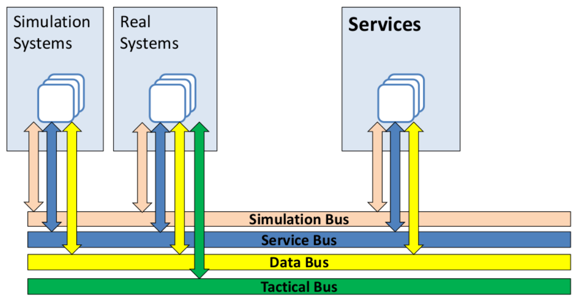

Figure 4: VIntEL Reference Architecture

- Deployable
- Maintainable
- Scalable
- Reusable: services can be used by multiple processes or composed into other services
- Independent units of business functionality: each service provides a business function that is independent of other services.
- Loosely coupled
- Platform-independent
- Based on open standards
- Low-Cost

3.1 SD VIntEL NATO Reference Architecture

MSG-131 final report on MSaaS: New Concepts and Service-Oriented Architectures [6] presents a number of reference architectures to use for MSaaS implementations.
Reference architectures are generic blueprints that may be used for deriving specific target architectures. When possible, building target architectures for specific simulation systems or simulation environments on foundations from established reference architectures will increase not only the efficiency of work in time and budget, but also the quality of the results, and will lead to improved interoperability.
The SD VIntEL reference architecture, depicted in figure 4, defines four buses for Data Exchange:

- Simulation bus, to exchange simulation data using HLA Evolved [14]
- Service bus, to connect with services using an Enterprise Service Bus.
- Data bus, for high volume/bandwidth data (such as camera streams or sensor data)
- Tactical bus, for C2 Interoperability.

It also classifies services into three different types:

- Domain Services: services that fulfil a specific task within the simulation
- Infrastructure Services: services used to control the infrastructure
- Adaptor Services: services that connect features outside the infrastructure

SD VIntEL has been used as a high level reference for the target architecture implementation, however significant deviations have been taken from the reference design.

3.2 CMRE Distributed Simulation Framework - Target Architecture

The target architecture, design fully embeds the recommendations contained in the MSG-131 final report on MSaaS [6] on System Design (SD) and on Simulation Environment Data (DA) and in particular:

- SD-1: Design and Document for Interoperability
- SD-2: Design and Document for Modularity and Composability
- SD-3: Favour Open Standards
- SD-4: Design for Securability
- DA-1: Enforce “Single Source of Truth” Principle

As depicted in figure 5 it has been designed in order to create a distributed simulation framework extending the work previously done in CMRE on interoperable M&S applications [3] [5].
The main feature of this architecture is that the simulator engine is structured as a Web service and is completely decoupled by the GUI that is optional and could be provided as a Web GUI complemented, in case of virtual simulation applications, by a Virtual-Reality Console.
The result is a flexible, modular, scalable and interoperable architecture that could be easily reused and expanded to quickly respond to upcoming requirements.
The architecture of the simulation engine is composed by the following functional blocks:

- Communication buses
- Time Manager and Real Time Clock
- Event Generators
- Discrete Event Processor
- Simulation Engine Core
- Locator
- Storers
- Rest Control Interface
- Services Facade

3.2.1 Communication buses

The target architecture uses four communication buses as the VIntEL architecture:

- Service bus, based on HTTP to connect with RESTful Web services. Ideally the simulation engine, should be connected only to the Service bus, delegating to specific Web services the role of interfacing with the other buses in order to encapsulate specific bus functionality into well defined, reusable Web services. However, mainly for a performances matter this is not always feasible.
- Simulation bus, to exchange simulation data using HLA Evolved [14]
- Tactical Data Link (TDL), using Over The Horizon - Gold (OTH-Gold)[26])messages over TCP for C2 Interoperability

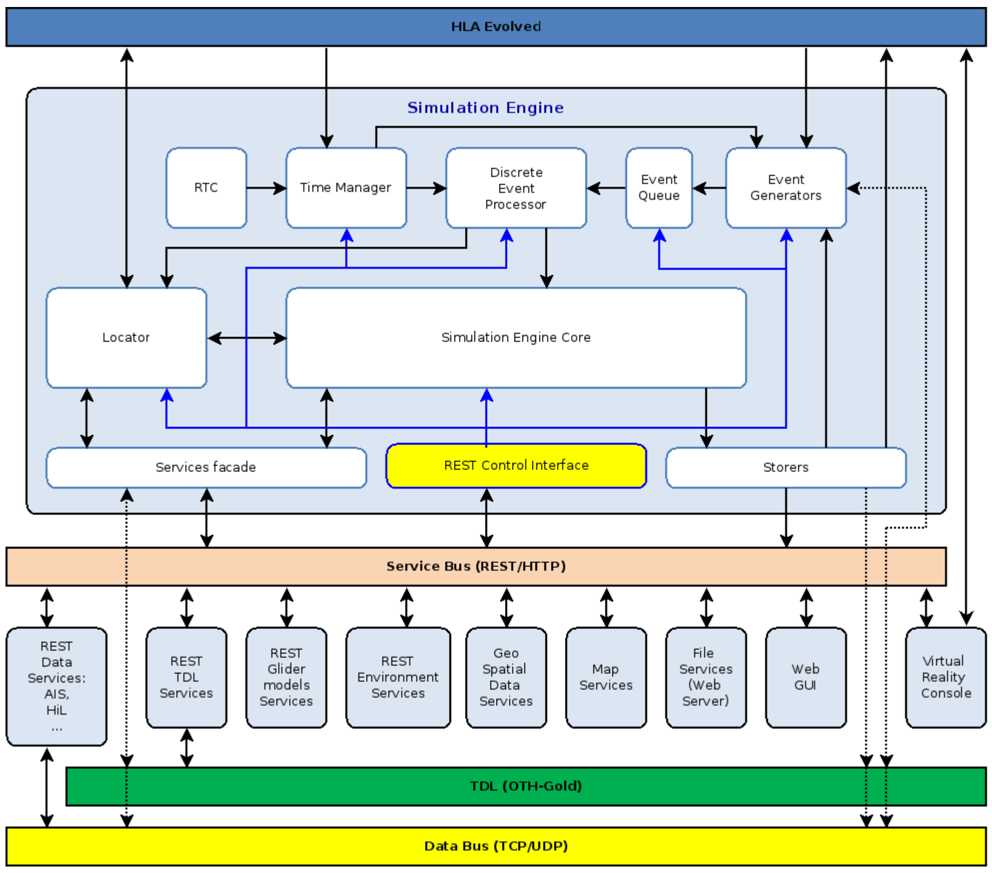

Figure 5: CMRE Distributed Simulation Framework - Target Architecture

- Data bus, TCP/UDP for high volume/bandwidth data (such as streaming sensors data)

Service bus: considering that HTTPs transport provides an adequate level of security for the type of application the Service bus has been implemented using RESTful Web services over HTTP/HTTPs rather than implementing a much more complex and cumbersome Enterprise Service Bus (ESB) solution as in the VIntEL architecture. An alternative to RESTful Web services is to use the more structured and secure SOAP based Web services but, REST has been preferred because of its lightweight and better performances.
Simulation bus: it is using the standard IEEE 1516-2010 known also as HLA Evolved[14]. Ideally HLA connectivity should be provided as a Web service, however it was estimated that, due to the intense interactions between the federate and the RTI (due to time management message exchange, object attributes updating and interactions firing), the HLA/REST conversion would have resulted in a sensible degradation of performances. For this reason the simulation engine is directly connected to HLA rather than through an HLA Web service.
Tactical Data Link bus: it is based on the OTH-Gold[26] protocol, using TCP/IP as transport. OTH-Gold provides a standardized method for transmitting selected data between C2 Systems. It is the primary message format for Tactical Data Processor (TDP) exchange. It is designed to be easily man readable for the non specialized user. Other, more powerful, but also more complex Tactical Data Link systems exist, such as the Link 11/16/22, but are not considered by the current architecture. They could however be included, when requested, by adding appropriate interface modules. Interface to the TDL is done by a specific Web service. The simulation engine connects to such service to subscribe to Tactical Data feeds available on the TDL bus. The TDL Web service then forwards data feeds on the TDL bus to the simulation engine by using a specific Application Programming Interface (API) provided by the simulation engine REST Control Interface.
Data bus: relies on a TCP/IP transport to interface with Real Systems. For example the Automatic Identification System (AIS) that provide streaming data on commercial shipping traffic.
As for the TDL bus, specific Web services would normally interface with the Data bus, however, in case of special requirements a direct connection to these buses from the simulation engine is always possible (dashed lines).

3.2.2 Time Manager and Real-Time Clock

The Time Manager manages simulation time in multiple ways:

- Real-Time: events are executed at the same speed of a Real-Time Clock (RTC),

- Accelerated-Time (2x, 3x...) is as the Real-Time case except that the Real-Time Clock output is accelerated by a factor.
- Fast-Time: events are processed as soon as possible, the Time Manager doesn't respect the interval between two events, but just execute the next event available in the queue as soon as the previous one is completed

Time Manager supports HLA time regulation to provide synchronization of federates progress.
HLA Federates may be designated as regulating federate. Regulating federates regulate the progress in time of federates that are designated as constrained. In general, a federate may be "regulating", "*constrained*", "*regulating and constrained*", or "*neither regulating nor constrained*". By default, federates are neither regulating nor constrained.
The RTI recognizes every federate as adapting one of these four approaches to time management. A federation may be comprised of federates with any combination of time management models. That is, a federation may have several federates that are regulating, several federates that are constrained, or several federates that are regulating and constrained.
A federate that becomes "*time regulating*" may associate some of its activities (e.g., updating instance attribute values and sending interactions) with points on the federation time axis. Such events are said to have a "*time-stamp*". A federate that is interested in discovering events in a federation-wide, time-stamp order is said to be "*time constrained*". The time management services coordinate event exchange among time-regulating and time-constrained federates.
Real-Time application with Man in the Loop, e.g. a mission control console to set the gliders missions, turn the simulator into a virtual simulator.
Time Manager is responsible of triggering the Discrete Event Processor.

3.2.3 Event Generators

The Event Generators is an aggregation of asynchronous (living on different threads) event generators which specialize the `EventGenerator` interface as visible in figure 6.
The available event generators are:

- `HLAEventGenerator`: generates events when objects attributes updates or interactions are fired by a federate on the HLA bus
- `TDLEventGenerator`: generates event upon receiving data on the Tactical Data Link
- `RealTimeEventGenerator`: generates events upon receiving data on the Data Bus
- `HiLEventGenerator`: generates events upon receiving data from the AUV hardware (either though the Data Bus or the HLA)
- `LocalEventGenerator`: generates events from the Simulation Engine Core
- `ScriptedEventGenerator`: generates a predefined set of events from a script file, an example of this could be a predefined list of waypoints to follow at given times, or other scripted information such as behaviours changes, alarms, HW failures...).

Multiple event generators can be active at the same time. Changing the configuration of the event generators allows to match the behaviour of the simulator to the intended use, e.g. as constructive simulator providing information to a decision support suite or as a virtual simulator with Man in the Loop and/or Hardware in the Loop for personnel training and/or HW/SW testing and validation purpose.
Event generators generate events asynchronously. Generated events are pushed into a synchronized (thread safe), time ordered queue so that they can be consumed by the Discrete Event Processor.

3.2.4 Discrete Event Processor

The Discrete Event Processor is designed to process events that multiple, asynchronous event generators push in a queue. Events in the queue are ordered by their time-stamp, not by arrival order, which is not equivalent since events generation time can in general be different from the time they are supposed to be executed (time-stamp).
The Discrete Event Processor is triggered by the Time Manager. At time ticks, provided by the Time Manager it processes timed events using a three-phase approach[25]: in the first phase it jumps to the next chronological event in the queue occurring at the time provided by the Time Manager; in the second phase it executes all events that unconditionally occur at that time (B-events); finally, in the third phase, it executes all the events that conditionally occur at that time (C-events). For maximum performances events can be processed concurrently applying a *Parallel Discrete Event Simulation*[24] approach .
Timed events can be of multiple types such as:

- model-update events, high frequency events that cause an update of the position, models and internal status
- federation-update events, coarser events that cause the Simulation-Engine to fire object attributes updates to the federation
- external events, coming from outside, such as the change of behaviours or mission, the order to release emergency ballasts, the turning on/off of sensors, the occurring of radio transmission, collisions with external assets or bottom structures, commanded hardware/software failures, events, not related to gliders that could be of interest by the simulation community such as ammunition detonation/hit/miss, radio transmission disruption...
- internal events such as collisions with the bottom, pressure alarms, low battery alarms, surfacing/diving of the glider
- timed events such as alarms or specific actions to be executed at a predefined time

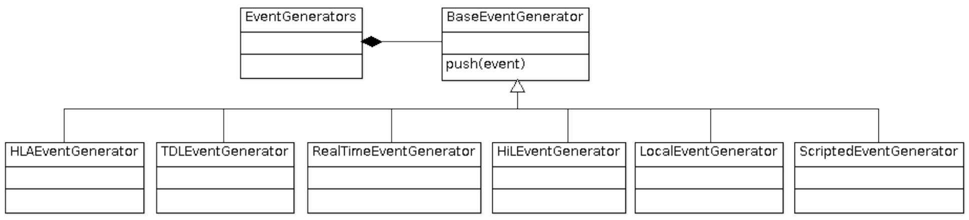

Figure 6: EventGenerators is an aggregation of EventGenerator objects

Events can be inserted with any orders, however they are always executed in order of time-stamp
Events are dispatched to the underlying Simulation Engine Core which updates the simulation status using a combined Discrete/Continuous models approach: the behaviour of the model is simulated by computing the values of the state variables at small time steps while the values of attributes and global variables is calculated at coarser event times.

3.2.5 Simulation Engine Core

The Simulation Engine Core calculates and stores the status of the simulator. It is event driven and uses the underlying models and the event information to update the status of the simulated unit e.g. last position, battery level, sensors/actuators status, HW/SW failures, alarms, damage level, collisions...
UML structure of the Simulation Engine Core and relationship with the Event Generators and the Discrete Event Processor is visible in figure 7.
The Simulation Engine Core is triggered by events received from the Discrete Event Processor and it may in turn generate events (e.g. diving/surfacing, collisions with the Sea bottom, depth/pressure alarms, HW/SW failures....) and dispatch them back to the Event Queue or to the buses through the Storers interface.
Events are dispatched to HLA in form of *HLA interactions*. Updates of the internal status are published to HLA through the Storers interface as *HLA objects attributes updates*.
Events and attribute updates may also cause information dispatching on the TDL or the Data Bus such as platform position updates to C2 or other real systems (e.g. internal status information, sensor performance predicted data calculated by underlying models...)
It has to be noted that The Simulation Engine Core doesn't calculate the location of the platform since this is the main responsibility of the Locator model.

3.2.6 Locator

The Locator model is responsible of calculating the six degrees of freedom location of the simulated platform along with it's bearing and speed. It is implemented as an interface. Multiple implementation of such interface are available, depending on the wished use/application of the simulator. However only one locator instance, per simulator, may be active at the time. The implementations are:

- HLA locator: interface with HLA to provide the location, speed and bearing calculated from an external dynamic/kinematic simulator engine.
- TDL locator: receive surfacing points or last known coordinates of the glider from the Tactical Data Link and uses its internal models to compute the coordinates of the glider (dead-reckoning) in between two known points or to predict the most probable surfacing point given the last know diving point. It works either in Real-Time and in Fast-Time.
- Real-Time locator: it works as the TDL locator except that receives the platform coordinates from the glider when surfacing through a TCP/IP dedicated connection.
- Hardware in the Loop (HiL) locator: interfaces directly with the glider sitting on the bench, to test the glider motion control algorithms (front-seat) and mission programmed beahaviors (Back Seat) complementing and extending the work previously done at CMRE within the context of the Persistent Autonomous Reconfigurable Capability (PARC) project with the Sparus II AUV reported in [3] and as in such implementation it may as well include an HLA interface as a bridge between the Robotic Operating System (ROS) of the AUV and the HLA federation: while the glider believe to be navigating in the ocean, its actuators control values are sent to this locator that uses them to estimate the glider position in the 3D environment, returning to the glider the simulated sensor readings (e.g. compass value, inertial navigation units values and water pressure). In this type of simulation the human glider pilot, controlling remotely the glider from the glider mission control room is as well included in the loop and the simulator can be used either to train the glider pilot as in a real mission but with the glider safely operating in the workshop bench.
- Glider Simulator locator: making use of the dynamic and kinematic models of the glider, reproduce the glider motion in the complex ocean environment. It also employs models to simulate the glider motion control algorithms (front-seat) and the

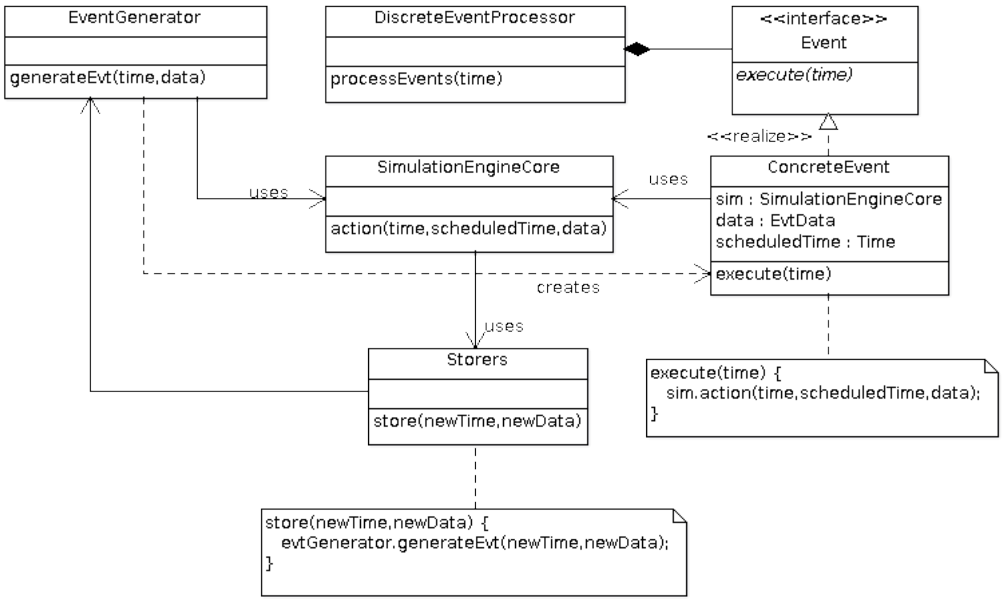

Figure 7: The Simulation Engine Core is event driven and implements a “Command” design pattern

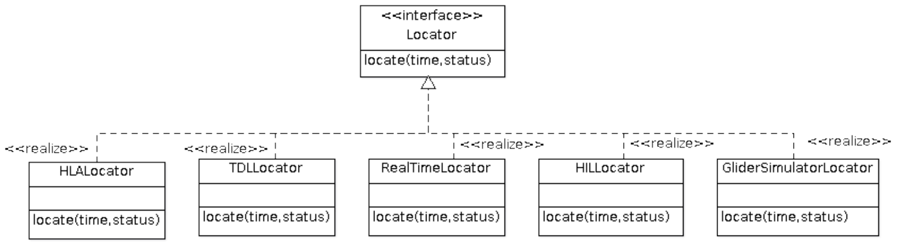

Figure 8: Locator UML diagram

glider mission programmed behaviours (back-seat).

The interchangeable Locator model is a flexible mechanism to adapt the locator to the application requirements. By changing the Locator model the Simulator may behave either as a Constructive Simulator providing information to a decision support suite or as a Virtual Simulator with Man in the Loop and/or Hardware in the Loop for personnel training and/or HW/SW testing and validation purpose.

3.2.7 Storers

The Storers module is in charge of the output of the Simulator and follows recommendation SD-3: Favor Open Standards of the MSG-131 final report on MSaaS [6].
Output data is provided in a variety of formats, however open standards are preferred to favor interoperability. For example the glider position is provided in the KML format to be displayed by any mapping/chartographic system compatible with such format (e.g. GIS software or Google Earth), in HLA format to interface with the majority of NATO Simulators and in the OTH-Gold Tactical Data Link format for compatibility with the majority of NATO Command and Control systems.
Other open data formats provided include the Open Geospatial Consurtium (OGC) data formats, NetCDF, GeoTIFF and the Matlab. HTML dynamically generated Web pages could be also provided.
Storers is a composition of classes which specialize the `Storer` interface as visible in figure 9. A storer is a module that dispatch information received from the Simulation Engine Core onto an output channel. Different storer models specialize the behaviour of the `Storer` interface.
Storers are called in cascading using a flavor of the Chain of Responsibility design pattern, so that only the storer responsible for a certain type of information will be able to forward such information toward the appropriate channel. A channel may be one of the buses, the internal queue or even a file such as a log file, or a specific format file, such as an HTML page to be displayed by a Web browser or a Matlab file to undergo further processing in an external tool.

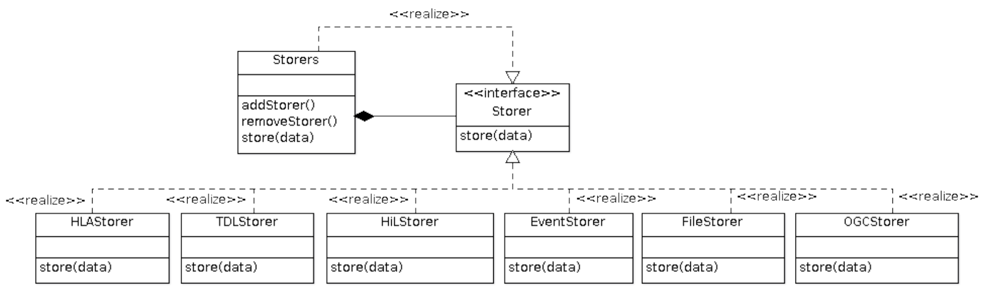

Figure 9: Storers is a composition of Storer objects

The available storers are:

- `HLAStorer`: send objects attributes updates (such as the glider location) or interactions to the HLA federation as a consequence of an internal status change or a generation of an internal event (e.g. a collision with the bottom, a HW fault, an alarm such in case the maximum depth is exceeded...)
- `TDLStorer`: send own track position to the Tactical Data Link
- `HiLStorer`: provide simulated sensors output to the HiL (either though the Data Bus or the HLA)
- `EventStorer`: generates events received from the Simulation Engine Core and sends them to the Event Generator to be inserted in the queue
- `FileStorer`: produces various types of files (e.g. HTML, KML, Matlab, JSON, CSV, XML, NetCDF, GeoTIFF...) and pushes the to a local Web server, available for download from a client application such a Web browser.
- `OGCStorer`: its an extension of the FileStorer that pushes geo-spatial files (e.g. NetCDF GeoTIFF...) to the Geospatial Data Service that in turn will provide them to a client application in the form of as OGC Web services such as Web Feature Services (WFS) or Web Map Services (WMS).

3.2.8 REST Control Interface

This is the REST front-end interface of the simulation engine. It makes the simulation engine available on the Services bus just as any other REST service. API available in the REST Control Interface include functionality to for:

Simulation instances management

- Create a new simulation
- List created simulations
- Destroy a created simulation
- Join a simulation

Simulation execution management

- Start simulation
- Pause simulation
- Stop simulation

Simulation status and parameters editing management

- Get simulation status
- Edit simulation parameters (e.g. real-time/fast-time, montecarlo runs, file/services outputs...)
- Edit models and environment parameters

Feed data from external sources

- Push in data e.g. coming from subscribed services (TDL, AIS...)

REST Control Interface runs in an Independent Process from the simulation engine. In this way it may create or destroy instances of a simulation engine as required.

3.2.9 Services Facade

The services facade module provides a unified interface to the set of underlying services defining a higher-level interface that makes the subsystem easier to use. This module hides the complexities of the larger system and provides a simpler interface to its clients: the Locator and Simulation Engine Core. It includes a single wrapper class that contains the set of member methods required by the clients. These members access the underlying services on behalf of the clients, hiding the implementation details.

3.3 Web Services

The glider models, environmental services and data services required by the simulator engine are provided in the form of Web services. For optimal performances and since the transport layer security provided by HTTP/HTTPs is considered to be satisfactory, RESTful Web services are used.
Basically the types of services that can be found are:

- Glider models such as the glider dynamic and kinematics, front-seat, back-seat, sensor models...
- Environmental services (bathymetry, climatology and weather forecasts)
- Data services to connect with external systems (e.g. Hardware in the Loop connection services; Automatic Identification System (AIS), that returns navigation information of maritime commercial traffic...)

- Geospatial Data services
- Map services

Some of these services have been implemented from scratch, some others are Open Standard services such as the Geospatial Data services or the Map services, based on Open Source Projects (e.g. GeoServer, Thredds or Mapnik).

4 Web services architecture

Web services architecture is represented in figure 10: a REST front-end servlet (an application, running on the server side, capable to respond to HTTP requests) written in Java and running into a Servlet container such as the Apache Tomcat represent the service front-end; it is connected through a Remote Method Invocation (RMI) or a Remote Procedure Call (RPC) to one ore more Micro-Services.

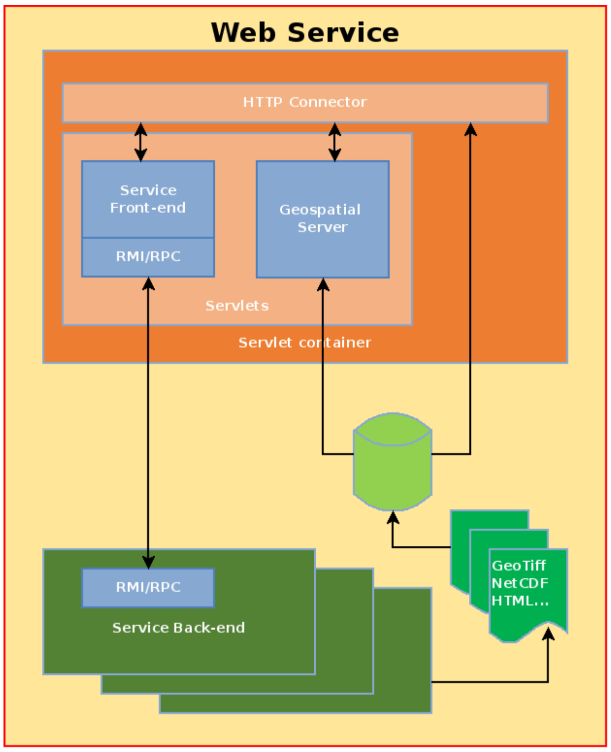

Figure 10: RESTful service architecture

Micro-Services may return directly their result to the Front-end, that would return it to its client or in alternative, in case of GeoSpatial data, they could publish it onto a Geospatial Data Server (such as Geoserver of the Open Source Geospatial Foundation or the Unidata Thredds), returning to the client the Unified Resource Locator (URL) of the published data. In such way the data becomes available to the Client using the OGC standards such as WMS or WFS.
REST is a good compromise to create lightweight, loosely coupled, language and operating system independent, distributed architectures. Behind the REST front-end a number of Micro-Services, written in different languages, represent the back-end of the service.

For example a mix of Matlab and Python processes would provide the glider mathematical models.
Interconnection between the Java front-end and the back-end of the service could be implemented using a number of different technologies:

- for C, C#, C++ (on POSIX systems), D, Delphi, Erlang, Go, Haxe, Haskell, JavaScript, node.js, Ocaml, Perl, PHP, Python, Ruby, Smalltalk backends, the Open Source Apache Thrift[21] framework for cross-language remote Procedure Calls (RPC), developed at Facebook to expedite development and implementation of efficient, scalable, cross languages services, has been highlighted as the optimal choice.
- when the back-end is Java, the most appropriate solution is the Java Remote Method Invocation (RMI). Apache Thrift represents however a suitable alternative.
- Matlab back-ends could be integrated with the Java front-end using the Matlab Compiler System Development Kit (SDK) which provide a Java library to interface directly with the Matlab Runtime engine.

In addition to the simulator engine a number of supporting Web services is included in the target architecture.

4.1 Environment Services

Environmental data services provide the simulation scenario with the following information:

- Bathymetry
- Climatology: historical seasonal/monthly average sea water temperature and salinity
- Weather forecast: sea state, waves height/direction, wind, rain, water salinity and temperature

Environmental services can be used either to: reproduce the conditions found during a real mission using recorded historical data; to plan a new mission, using either forecast data or typical seasonal data (climatology); and also to explore what would happen by applying or superimposing certain conditions on the scenario area (e.g. forcing a sea state or wind direction, applying a current field or introducing an anomaly in the salinity of the water at a certain depth that could severely impact on the navigation capabilities of the glider).

4.2 Glider models

Include a number of services that implement glider models or models used by glider models described in Chapter 2. The simulator aim also to become a test-bed and tuning tool for dynamic and kinematic gliders models.

4.3 File Services

Files produced during the simulation execution, such as HTML, KML, Matlab, JSON, CSV, XML, NetCDF, GeoTIFF are pushed to a local Web server and become available for download from a client application such a Web browser and can be further processed by external tools.

4.4 Geo Spatial Data Services

Geo-spatial files (e.g. NetCDF GeoTIFF...) are provided by the Geospatial Data Service in the form of OGC Web services such WFS or WMS.

4.5 Map Services

Map Services are used to provide map tiles to a mapping application. Such services are used by the Web GUI to display the map with the geo-referenced positions of the Glider. The availability of a map-service prevent the need to connect to an online map data service provider making the simulation engine independent from the availability of an internet connection.

5 Notes about securing the system

Exposing Web services to Wide Area Networks such as the Internet requires that a number of precautions are taken against malicious attacks. NATO accreditation must be obtained before the service is made available on the Internet. The main aspects of the security accreditation are based on the NATO Communication and Information Agency (NCIA) requirement to follow the Open Web Application Security Project (OWASP) Application Security Verification Standard including:

- Authentication
- Session Management
- Access Control
- Input Validation
- Cryptography
- Error Handling and Logging
- Data Protection
- Communication Security
- HTTP Security
- Business Logic
- Files and resources
- Miscellaneous requirements

The original architecture shown in figure 10 was subsequently expanded to that shown in figure 11. It may be seen that the original architecture now forms only a sub section of the revised architecture. In particular the revised architecture includes firewalls, Web server, Service Provider, Identity Provider, Directory Server and a users database.
A user that need to access a service connects to the Service Provider hosted on the Web server in the Demilitarized Zone (DMZ), the area comprised between the two firewalls in which no data or software is stored except the Web server own log files. If the user has not been already authenticated it is redirected to the Identity Provider. The Identity Provider performs the user Authentication checking the supplied credentials against those stored in the Users Database through the Directory Server (usually a Lightweight Directory Access Protocol (LDAP) service). Once authenticated the user is redirected to the proper Web service. This process is further filtered according with the Authorizations that each user owns that are contained in the Users Database.
Confidentiality and Integrity of the data are granted on the Client side by the HTTPs transport protocol. Firewalls filter the input/output traffic. The Web server in the DMZ acts as a secure front-line for the incoming requests redirecting clients traffic to the back-standing Web services. Communication between the Web server and the Web services could be done using the lighter and faster HTTP protocol (with no encryption) however communication on the client side is always protected by HTTPs TLS/SSL encryption.
Such architecture allows enforcing the security requirements required to undergo the accreditation process that includes the following phases (extracted from Communication and Information Services (CIS) Security Technical and Implementation Directive for the Security of NATO's Internet Presence):

- Security Level Review
- Design and Architecture Review
- Vulnerability Scans
- Health Checks
- Code Walk-through
- Application Penetration Testing
- Code Review
- Create and Review UML Models:
- Create and Review Threat Models:
- Test Configuration Management

6 Graphical User Interface

The GUI is decoupled from the simulation engine and is optional: for applications where the simulation engine is used as a computation engine the GUI may not be used at all and the hosting application would just interface to the simulation engine directly.
For end user applications where a Graphical User Interface is required a Web application is the optimal solution. Web applications do not need to be installed or configured. The user just need to know the URL address of the application to be able to use it.
The main components of the Web GUI are:

- Simulation Management Controls: to create, list, join or destroy a simulation.
- Simulation Time Management: to start, pause, stop the simulation execution
- Simulation status: display the simulation status in various forms: tables, plots, graphs...
- Simulation parameters editing: to edit simulation parameters
- Simulation models editing: to edit the models parameters
- Map View: to display a Map with the simulator graphical output data overlaid

Web GUI components may be combined into different views. Authorizations provided by an external Identity Provider define which controls and features are available to authenticated users.

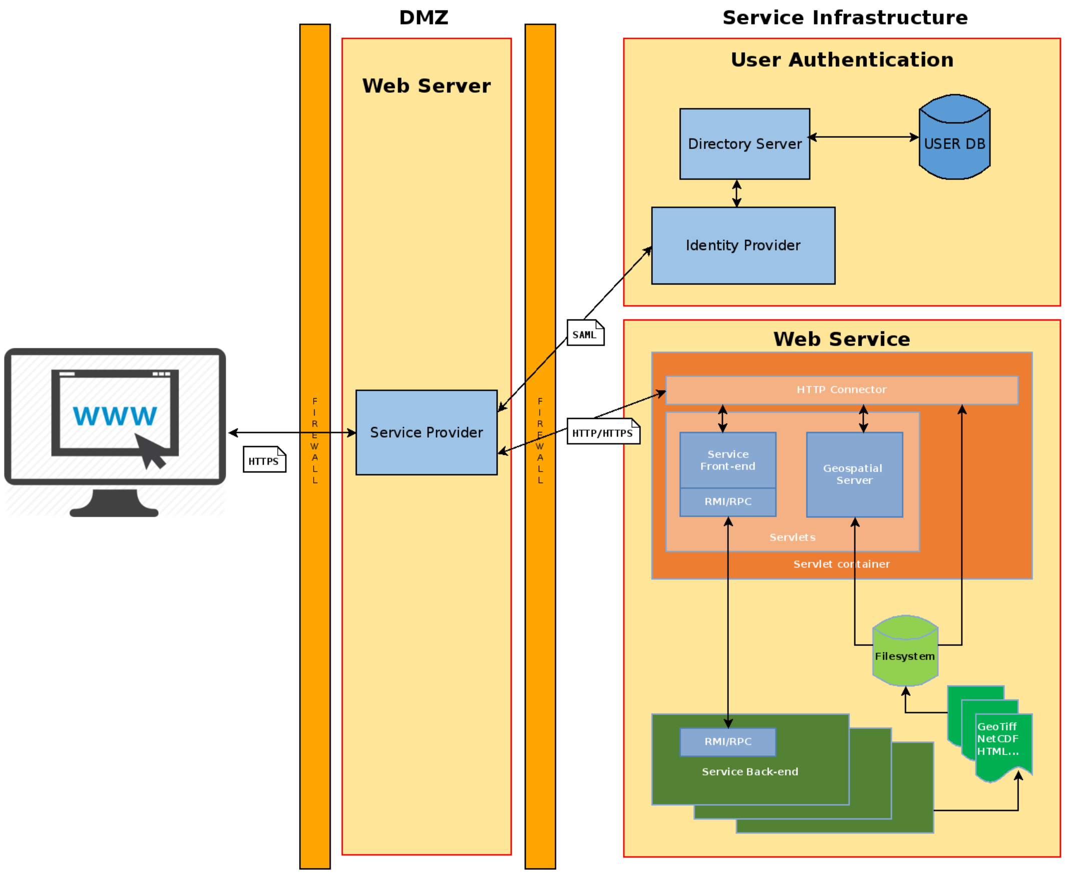

Figure 11: Securing Web Services

6.1 Virtual-Reality console

For Virtual-Simulation use cases, a standalone application with HLA interface represents the most natural solution but moves a little bit out of the MSaaS concept since it would not be provided anymore as a service in the more strict sense of the term but would require the installation of the application package on the client terminal. On the other side, implementing the Virtual-Reality console as a Web application, in order to provide it as a service to the end user, would introduce aspects related to graphics and communication performances from within a browser that are not in the scope of this paper.
The Virtual-Reality Console stand alone application connects as a federate to the HLA federation and subscribe to the objects attribute updates and interactions of interest.
In case of multiple simulator federates, an additional federate would be required to manage the interactions between the federates, such as collisions between the simulation entities and/or static features e.g. an underwater structure.
To avoid interoperability issues between federates due to the use of uncorrelated environmental data, it is necessary to enforce the MSG-131[6] recommendation DA-1 Enforce Single Source of Truth Principle through the whole federation that requires that each environmental data source is unique and all application-specific data items or formats are derived from such source data item.

6.2 The Map View

The Map View allows to display geo-referenced output of the simulator onto a map. In figure 12 it is possible to see the Map View with the AUV icon in Mil STD 2525D[9] AUV icon and a sea velocity field WMS layer overlay.
Information that can be displayed as map overlay include:

- Platforms positions(e.g. the simulated glider icon or other entities received from HLA/TDL/AIS) along with their routes and tracks using the KML format
- environment data layers such currents, winds, waves, water temperature/salinity as WMS data layers
- Models/simulation produced data layers (e.g. sensor ranges, risk areas....)
- Navigation charts
- Elevation and bathymetric contours
- Weather forecast data

Additional layers can be displayed as WMS layers, vector layers (e.g. KML or GeoJSON) or as tiled layers.

References

[1] D. Pochazka, J. Hodiky, “Modelling and Simulation as a Service and Concept Development and Experimentation”, 2017 International Conference on Military Technologies (ICMT)

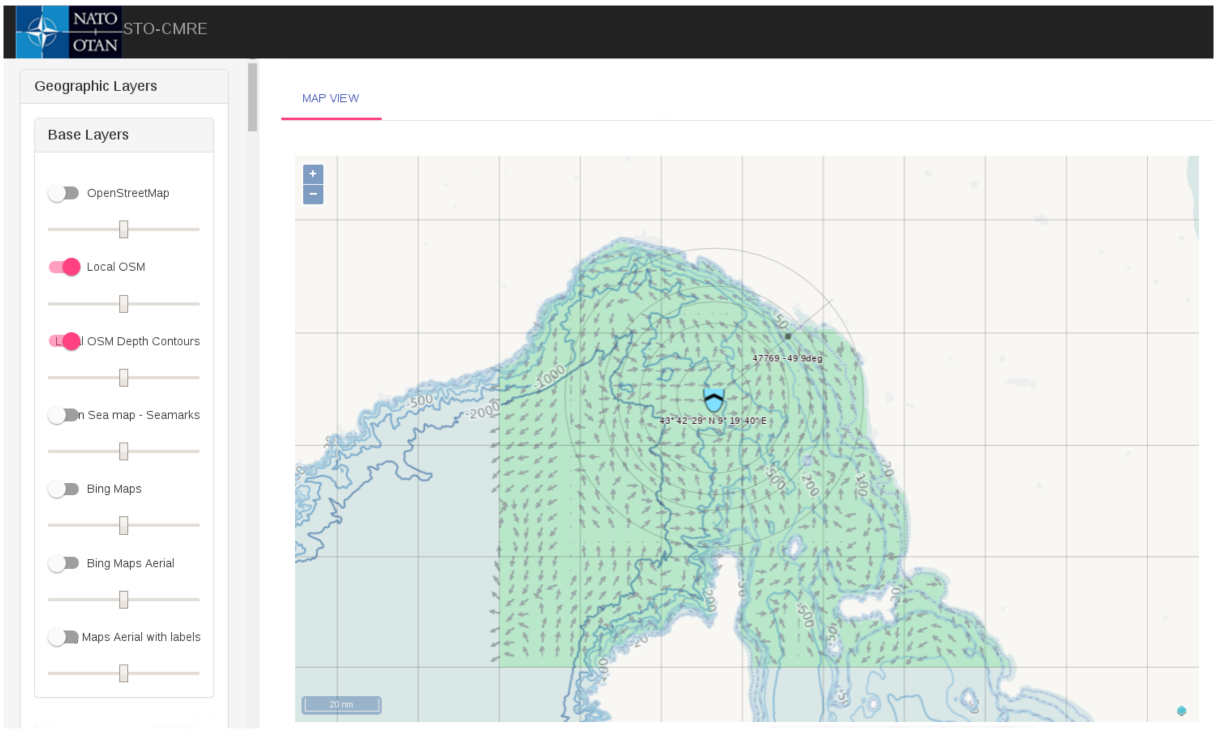

Figure 12: The Graphical User Interface MapView with a sea water velocity field and a Mil STD 2525D AUV icon overlayed

[2] U. Latifa, T. W. O. Putri, B R Trilaksono, E. M. I. Hidayat, “Modelling, Identification and Simulation of Autonomous UnderwATER Glider in Longitudinal Plane for Control Purpose”, 2017 2nd International Conference on Control and Robotics Engineering

[3] A. Carrera, A. Tremori, P. Caamao, R. Been, D. C. Pereira, A. G. Bruzzone, “HLA interoperability for ROS-based autonomous systems”, 2016, International Workshop on Modelling and Simulation for Autonomous Systems (pp. 128-138). Springer International Publishing.

[4] “Operational Concept Document (OCD) for the Allied Framework for M&S as a Service: DRAFT v0.28”, 09 November 2016

[5] A. Bruzzone, A. Tremori, M. Oddone, D. Crespo Perreira, M. Scapparone, S. Vasoli, M. Corso, I. Piccini, “HLA based interoperable simulation: MCWS-MSTPA federation”, La Spezia: CMRE, 2015/01. (CMRE Memorandum Report (MR series) CMRE-MR-2015-002

[6] Final Report of Specialist Team MSG-131, “Modelling and Simulation as a Service: New Concepts and Service-Oriented Architectures”, 2015, STO Technical Report

[7] NATO STO: Final Report of NATO MSG-086 “Simulation Interoperability”, STO Technical Report, STO-TR-MSG-086-Part-I, January 2015.

[8] “Modelling And Simulation Architecture Standards For Technical Interoperability: High Level Architecture (HLA)”, 2015, North Atlantic Treaty Organization

[9] “MIL-STD-2525D”, 10 June 2014, Department of Defense interface Standard, Joint Military Symbology

[10] E. Cayirci, “Modelling and Simulation as a Cloud Service: a Survey”, Proceedings of the 2013 Winter Simulation Conference

[11] K. Isa, M. R. Arshad, “Buoyancy-driven underwater glider modelling and analysis of motion control”, Indian Journal of Marine Science, Vol 41(6), December 2012

[12] C. Wang, A. Anvar, “Modelling and Simulation of Motion of an Underwater Robot Glider for Shallow-water Ocean Applications”, World Academy of Science, Engineering and Technology, International Journal of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering, Vol:6, No 12, 2012

[13] R. Grasso, D. Cecchi, M. Cococcioni, C. Trees, M. Rixen, A. Alvarez, C. Strode, “Model based decision support for underwater glider operation monitoring”, OCEAN 2010 MTS/IEEEE, Seattle

[14] “IEEE Standard for Modelling and Simulation (M&S) High level Architecture (HLA) Framework

and Rules", IEEE std 1516-2010 (Revision of IEEE Std 1516-2000), pp. 1-38, 2010.

[15] B. Garau, M. Bonet, A. Alvarez, S. Ruiz, A. Pascual, "Path planning for authonomous underwater vehicles in realistic oceanic current fields: application to gliders in the western Mediterranean sea", 2009, Journal of Maritime Research Vol. VI N. II

[16] K. Al-Zoubi, G Wainer, "Using REST Web-Services Architectures for Distributed Simulation", 2009, ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulations

[17] M. Quigley, K. Conley, B.P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, n. Andrew, "ROS: an open-source Robot Operating System", 2009, ICRA Workshop on Open Source Software

[18] S. Sarkka, "Unscented Rauch–Tung–Striebel Smoother", 2008, IEEE Transactions on Automatic Control (Volume: 53, Issue: 3, April 2008)

[19] O. Schofield, et al, "Slocum gliders: robust and ready" Journal of Field Robotics, 24(6), 473-485 (2007), Wiley InterScience.

[20] "NATO Consultation, Command and Control Board (NC3B): NATO Architecture Framework", Version 3, AC/322-D(2007)0048-AS1, 2007

[21] M. Slee, A. Agarwal, M. Kwiatkowski, "Thrift: Scalable Cross-Language Services Implementation", 2007, Facebook, 156 University Ave, Palo Alto, CA

[22] Bhatta, Pradeep, "Nonlinear stability and control of gliding vehicles", 2004, Diss. Princeton University

[23] R.T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures", 2000, UNIVERSITY OF CALIFORNIA, IRVINE, Dissertation

[24] A. Cassel, M. Pidd, "Distributed discrete event simulation using the three-phase approach and Java", 2001, Simulation Practice and Theory 8 (2001) 491-507

[25] M. Pidd, "Computer Simulation in Management Science", 1998, ISBN : 978-0-470-09230-9

[26] "Operational Specification for Over-The-Horizon Targeting Gold", Revision D, OS-OTG (Rev D), 1 September 2000, published under the direction of CNO (N62) by Navy Center For Tactical System Interoperability

[27] T. Buckman, "NATO NETWORK ENABLED CAPABILITY FEASIBILITY STUDY EXECUTIVE SUMMARY", Version 2.0, 2005, NATO Consultation, Command and Control Agency