IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 18, 2018, accepted July 30, 2018, date of publication August 30, 2018, date of current version September 21, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2865683

Hybrid Simulation and Test of Vessel Traffic Systems on the Cloud

MASSIMO FICCO[1], (Member, IEEE), ROBERTO PIETRANTUONO[2], (Senior Member, IEEE), AND STEFANO RUSSO[2], (Senior Member, IEEE)
[1]Dipartimento di Ingegneria Industriale e dell'Informazione, Università degli Studi della Campania Luigi Vanvitelli, 81031 Caserta, Italy
[2]Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione, Università degli Studi di Napoli Federico II, 80125 Naples, Italy

Corresponding author: Roberto Pietrantuono (roberto.pietrantuono@unina.it)

This work was supported by the GAUSS National Research Project through the MIUR Program under Grant PRIN 2015.

ABSTRACT This paper presents a cloud-based hybrid simulation platform to test large-scale distributed system-of-systems for the management and control of maritime traffic, the so-called vessel traffic systems (VTS). A VTS consists of multiple, heterogeneous, distributed, and interoperating systems, including radar, automatic identification systems, direction finders, electro-optical sensors, and gateways to external VTSs, information systems; identifying, representing, and analyzing interactions is a challenge to the evaluation of the real risks for safety and security of the marine environment. The need for reproducing *in fabric* the system behaviors that could occur *in situ* demands for the ability of integrating emulated and simulated environments to cope with the different testability requirements of involved systems and to keep testing cost sustainable. The platform exploits hybrid simulation and virtualization technologies, and it is deployable on a private cloud, reducing the cost of setting up realistic and effective testing scenarios.

INDEX TERMS Cloud computing, emulation, HLA, simulation, system-of-systems, testing, vessel traffic systems.

I. INTRODUCTION

A Vessel Traffic System (VTS) is a large-scale system whose objective is to support safety and efficiency of navigation and protection of the marine environment, adjacent shore areas, work sites and offshore installations from possible adverse effects of maritime traffic. It is a typical critical infrastructure system manufactured according to the System-of-Systems (SoS) paradigm, where multiple interoperable systems are interconnected by means of proper middleware solutions through local or wide-area networks [1]. A VTS is generally organized in three hierarchical levels, including local control centers, area centers, and head quarters. Each local VTS can integrate a wide variety of remote sensors and inter-operate with several external systems.

Due to their huge complexity and scale, setting up realistic and cost-efficient *testing* sessions for a VTS is a serious concern. A typical test scenario can consist of several VTSs hierarchically connected, with a multitude of physical and functional heterogeneous components across different technological domains (representing physical hard components as well as soft components), and human and organizational components, not necessarily belonging to the same entity or organization. The monitored area can include from 100 to 5,000 marine objects (including ships, buoys, offshore plant, etc.), as well as different identification base stations, weather stations, and huge number of radar, cameras, and direction finders, with the network scenario involving both geographical and local connections.

This complexity is likely to lead to unexpected (hence untested) behaviors, mainly affecting dependability and performance, that usually become evident only during systems operations on-site and, in particular, in presence of stress, unexpected conditions or events not observed until then. For example, during the monitoring of traffic navigation in a seaport, if not covered in the system specification, a ship out of the norm (i.e., longer than those specified by applicable regulations) could be identified by VTS as two distinct ships when it passes behind a buoy.

A failure in one component can propagate within the subsystem and to other subsystems, provoking cascading failures that can produce severe unpredictable consequences well beyond the impact zone. Diagnosis of malfunctions during system operation is a very difficult and expensive task, that often requires urgent actions on sites by specialized teams of engineers, with high cost and time overruns for the company.

Assessing the dependability of such a system before deploying it is as much important as difficult. The main challenge is to be able to reproduce ''relevant'' scenarios locally, in order to gain knowledge about the real behavior of the system in-factory as it would be on-site. It would require sophisticated modeling practices and expensive experimentation environments/infrastructures to simulate and test the functionality of the whole SoS and analyze the interconnections and interactions among its parts [3].

In the context of a public-private research project named DISPLAY (Distributed hybrId Simulation PLAtform for ATM and VTS sYstems) [4], we have considered the *hybrid simulation* as a valuable testing option, also known as pseudo-dynamic testing [5], coupled with a cloud-based deployment. Hybrid simulation can combine emulation and simulation together in order to cope with the extreme heterogeneity of testing requirements for the described systems. The simulation can be used to reproduce the behavior of the external systems of interest (e.g., sensors, radar, web cam, and external software COTS), whereas emulation can be adopted to reproduce the execution of the real target system. Deployment over virtual infrastructures and a service-oriented architecture provides flexibility and scalability, optimizing time and cost of testing and maintenance activities.

The article presents a platform to implement locally controlled testbeds (managed in factory) for VTSs, through the integrated use of distributed and hybrid simulation techniques. The platform supports synchronization and communication services needed to make heterogeneous distributed simulation tools and emulated components interoperable. Moreover, service interfaces are provided to configure and manage (at architectural level) the simulation scenario through the platform, which runs and manages the simulation process (i.e., emulate the system behavior), without the effort and cost needed for building and maintaining complex testbed infrastructures. By adapting the Platform-as-a-Service (PaaS) concepts to the simulation domain, the simulation framework can be offered as a service to testers on a cloud infrastructure (SIMPaaS), and exploited to build large-scale test scenarios.

The rest of the paper is organized as follows: in Section II, we survey potential solutions from the literature to simulate and test a critical SoS, such as VTS. Section III discusses the role and potential of simulation, hybrid simulation and simulation-as-a-service solutions for setting up testing environments of a SoS. Section IV describes the requirements of the testing platform for the VTS case. Section V and VI present our solution. Section VII reports about the testbed implementation and the main test scenarios of the target VTS. Section VIII discusses the cloud-based distributed simulation-as-a-service solutions proposed in the literature. Section IX discusses implications and concludes the paper.

II. SoS TESTING AND SIMULATION

A VTS is a prominent example of critical System-of-Systems. According to De Laurentis [6], a SoS consists of ''*multiple, heterogeneous, distributed, occasionally independently operating systems embedded in networks at multiple levels that evolve over time*''. This entails a inherent structural and dynamic complexity [7]. *Structural complexity* derives from: *(i)* a multitude of physical and functional heterogeneous components, across different technological domains, representing physical hard components (e.g., railway, radar, direction finder, etc.), as well as soft components (such as SCADA, information systems), and human and organizational components, in general not belonging to the same entity or organization; *(ii)* the scale and dimensionality of their connectivity and their geographical extension. *Dynamic complexity* manifests itself as emergence of unexpected system behaviors in response to changes in the environmental and operational conditions of its components, as well as of disruptive events, (e.g., natural events, criminal and malicious activities, market and policy factors, outages), introducing additional complexity in the management and control.

Identifying, understanding and representing such complexity is a challenge to the evaluation of weaknesses and vulnerabilities of subsystems in consequence of an initiating event. It requires sophisticated testing environments to support activities as: *(i)* in-factory integration and system testing without the need of expensive geographic-scale evaluation on-site; *(ii)* testing to assure the correct functioning of the product before delivery, to avoid expensive re-designs, late bug fixes and delivery of bad-quality products; *(iii)* definition of tuning actions before their first on-site execution; and *(iv)* verification of failure mitigation strategies.

Generally, complexity forces testers to simplify the testbed configuration to keep its realization cost low. For instance, some software components are assumed to work properly (e.g., operating systems), thus, software on the testbed is a simplified version of the operational one. Also, each scenario could take several hours to complete – thus, only test scenarios obtained from recording behaviors observed in-situ are considered. Scenarios taken from the operational phase assures to test representative cases, but focusing always on the same (few) scenarios does not allow evaluating new cases or specific, unforeseen, situations not included in the requirements, and very few cases can be explored carefully. Such simplifications greatly reduce the number of tests, but to the detriment of a deep quality assurance. The natural pay-off is a partial coverage of requirements and a risk of low representativeness of tested scenarios, and a consequent increased risk of poor quality of final products.

On the other hand, reproducing real complex scenarios in-factory for a deeper quality assurance is still very expensive and time-consuming (and often even unfeasible), as multiple independent and heterogeneous entities, with their own temporal dynamics, must be involved and synchronized among each other. In this direction, several simulation methods are proposed in the literature for vulnerability assessment and risk analysis of critical infrastructure systems (CISs):

- *Logical methods*, which include logic trees, Markov Chains, Markov/Petri nets, Bayesian networks, etc., are capable of capturing the logic operating of a system, and of identifying the combinations of failures of elements potentially leading to the system failure [9]. Drawbacks of these methods are the exponential growth of system configurations when the number of components and states increases, as well as the significant efforts in logic modeling and quantification.
- *Functional methods* include agent based model, system dynamic model, economic-based approaches; they are capable of capturing the dynamics of interrelated operations among hardware and software elements of a system and its interaction with the environment [3]. Nonetheless, it is difficult to calibrate the model parameters and validate complex models.
- *Structural/topological* methods represent CISs by networks, where nodes are the components and links are the physical and relational connections among them [11], [10]. Networks can be analyzed by simulation and analytical methods, which can capture relevant structural properties, identify critical components and support the improvement of system robustness. However, they hardly capture the dynamic complexity of real SoSs, and their computational cost can be prohibitive when components and links are modeled in detail.

There exist many different tools and frameworks that developers can use to build complex simulations upon. Some of these focus on specific application domains, others focus on domain-independent scope and support simulation in general. Examples of domain-specific simulation frameworks are *Mosaik* [12], which is a flexible Smart Grid co-simulation framework, and *RinSim* [13], a simulator for logistics problems. Currently, several specific VTS simulators have been proposed [15], [16]. On the other hand, they are not designed to test VTS, but only used to train personnel in handling marine traffic. Examples of general simulation frameworks are the *Jadex* project [17], which is a multi-agent based simulation framework, and *COSSIM* [14], a simulator specifically designed for providing accurate simulation of cyber physical systems. On the other hand, generic frameworks would force the developer to implement realistic simulation scenarios from scratch, and as the next sections will clarify, it might complex to capture heterogeneity and structural and dynamic complexity of a SoS, such as the VTS. However, no specific simulation platform has been proposed to test VTS.

III. VTS SIMULATION AS-A-SERVICE

The evaluation of risks, vulnerabilities and resilience of systems can be carried out through a simulation process, by modeling reality and quantifying defined metrics. In general, *Simulation* can be a valuable support to improve test representativeness and coverage at low cost. Being able to accurately simulate the behavior of a system allows engineers to drastically reduce the time and costs of its dependability evaluation, enabling more effective testing, analysis of alternative design decisions, identification of architectural bottlenecks, early detection of bugs, and so on. Nonetheless, due to heterogeneity, structural and dynamic complexity, the SoS representation and modeling is such that its characteristics cannot be fully captured and quantified in a reliable way, and large uncertainty is always present [8].

A simulation service for a SoS would present several serious challenges: various simulation tools, real subsystems (usually Commercial Off-The-Shelf) and experimental platforms need to interact in a coordinated way within a distributed environment. Their integration would require sophisticated modeling practices and complex experimentation environments. Also, despite the advantages of simulation, the complexity of systems and the large number of involved entities would still lead to very high cost and simulation time. To be effective, simulation should manage these complex aspects and be, at the same time realistic, time-optimal and cost-effective - objectives which contrast each other.

Hybrid distributed modeling strategies represent a viable alternative to design simulation platforms for testing. In hybrid simulation, the emulated parts are the system or components under test, e.g, the Vessel Traffic System (VTS) software, as well as each element that cannot be tested in real operating environment, for example a network router. The simulated parts can be, for instance, the sensors or objects present within the operating scenario, such as a radar, a ship in a port or a marine object, which can be used to generate the experimental workload or perturbation in the system operation. Other elements that can be simulated are components or subsystem that cannot be directly used, for example, because they are not accessible or available for the testing activity. Finally, the real systems are all the other subsystems that are not the target of testing, but from which it is possible to obtain, in real-time, the data streams needed to reproduce a real experimental scenario.

A. HYBRID SIMULATION AS A SERVICE

The high-level objective of hybrid simulation is to reproduce, in-factory, a high percentage of tests usually executed in-situ, in a much more efficient way. The VTSs targeted in this work are typical large-scale critical systems, made up of several systems geographically distributed in local and remote centers, which should interact through a communication infrastructure with other COTS systems (such as external software entities and sensors sites). Therefore, the simulation framework should be able to manage large-scale simulations with several hundreds of entities, typical of a mission-critical domain. To drastically reduce the setup cost of testing, testers should be able to easily setup scenarios without caring about details of the underlying infrastructure and platform, e.g., hardware and software configurations, communication networks, software licenses and upgrades, scaling, SLA-compliant QoS assurance. Therefore, hybrid and distributed simulation services, supported by novel

technologies for resources virtualization and working environment reproduction, represent the most promising option. This can be realized by designing the simulation platform on atop of a cloud computing infrastructure, namely, by implementing the simulation environment as a service [19].

In the literature, Simulation-as-a-Service (SIMaaS) model is conceived to provide access to simulation services on-demand, running in a shared pool of computing resources, at low cost and scale as needed [18]. By adapting this concept to the cloud computing domain, simulation could be built on Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). At SaaS layer, the simulation functionality is implemented as software and hosted in the cloud. All the simulation services are provided according to a service-oriented architecture approach. This approach unifies the invocation of different simulation types and allows combining different implementation models. In particular, a simulation service can be either directly used through thin clients (e.g., web browser), if testers require specific simulation functionality, or by other simulation services (as a sub-simulation) to implement complex simulation workflow [20]. PaaS abstract from the operating system level and provides a dedicated middleware programming interface to develop and run simulation applications on cloud without managing the underlying hardware and software layers. The platform automatically deploys the simulation components and starts new instances to achieve scalability [21]. However, it restricts the application types to those supported by the platform. At IaaS layer, it does not restrict the type of applications deployable on the IaaS, but it abstracts away only from the details of the physical hardware. Virtual resources are provided to the testers (e.g., the virtual machines of the system under test), allowing fine-grained control of the software stack, such as operating systems.

According to Preisler *et al.* [21], if the simulation is black-box and not implemented with explicit cloud distribution in mind (e.g., in the case of an already existing simulation system), it might be more feasible to use the IaaS approach. Instead, if the simulation is built from scratch or the simulation model is a simple one, it is more feasible to use PaaS to encapsulate the model within a simulation component that is implemented by the PaaS Application Programming Interface (API). Finally, if the simulation is offered as a service, SaaS paradigm can be adopted, in which simulation software is used as services through the cloud.

The test scenarios of a VTS – and of critical SoS in general – are characterized by a large number of distributed and heterogeneous components (from functional and operational environment point of view), interacting with other COTS systems by a complex network structure. Therefore, the use of cloud-based simulation as a service allows exploiting configurability, elasticity and tenacity offered by cloud computing. In the described context of hybrid simulation, it is necessary to emulate the real operation environment – therefore, fine-grained control of the software stack must be assured and a IaaS approach is more flexible. As for COTS and external components, such as radar, surveillance camera, and water systems, it could be only needed to simulate some behavioral patterns. Considering their complexity, the implementation of simulators from scratch could be excessively complex and time-consuming – thus, the platform has to offer facilities to reuse already existing system components simulated by dedicated environments, such as *Matlab, Modelica, OMNET++*, etc.. Encapsulating their functionality, namely *micro simulation functionality*, by a service simulation component would enable their execution in a unified way on a scalable cloud infrastructure [21]. However, in order to implement complex simulation processes, where several simulations are executed in parallel while exchanging data, the time model of different simulators must be synchronized. Therefore, specific *macro simulation functionality* must also be provided, such as communication and synchronization, to allow the interaction among different simulators and interoperability between the simulated and emulated environments. Although, the PaaS approach restricts the types of applications to those supported by the platform, it can be more suitable for realizing hybrid simulation as a service. Therefore, the simulated components can be new components built on top of the platform API, or existing simulation tools to encapsulate as a black box simulation within a component implemented by the API of the PaaS; but in this case, the simulation does not profit from the distribution, scalability and robustness properties of the PaaS platform.

Essentially, the proposed solution consists in implementing the simulation platform as a service (SIMPaaS), enabling at PaaS level both the macroscopic simulation functionality (for interlinked simulations) and the microscopic functionality of simulated components, as well as allowing fine-grained control of the software stack of emulated components at IaaS level. This solution abstracts, by virtualization, the use and management of physical resources and the synchronization and communication mechanisms of each involved simulation tools, which are very complex in the considered domain.

IV. SIMULATION PLATFORM REQUIREMENTS

In the following, the requirements of the simulation platform for the VTS scenario are outlined.

In general, a VTS is organized in three hierarchical levels, including local control centers, area centers, and head quarters. Each local VTS can integrate a wide variety of remote sensors (e.g., radar, electro optical sensor, automatic identification system, and direction finder), as well as external systems (e.g., long range identification systems (that is, satellite), tracking, weather and communication systems. In such complex scenario, hybrid simulation can support the process of integration and testing of such system, allowing to: (1) add and remove dynamically sensors; (2) simulate a failure of a component; (3) simulate accidents; (4) simulate communication degradation over a large scale wide area network; (5) control the number, type, velocity, and trajectory of the ships; (6) modify the number of hosts composing the VTS; (7) generate predefined traffic flows. In particular,

the traffic flows soliciting the system are not a variable that can be controlled in real-world situations, so that, knowing the behavior of the system when these variable parameters change is a kind of a guess that conversely can be carefully studied in simulated systems.

An example of test scenario is the "*Degraded AntiSplitting Tracking*" test. The objective of this test is to check if the *anti-splitting tracker* works correctly: whenever an object exceeding the expected size is detected by the radar, the system should still identify it as one single big object rather than two smaller objects (i.e., splitting the object). This is expected to work even under degraded performance up to 50% of bandwidth loss. Specifically, the test must replicate a critical situation where a radar has to manage the track fusion correctly. One single big target can generate track splitting (because of the size and/or because its slipstream), that can lead to multiple tracks for the object. This scenario is emulated both under nominal network conditions, in which case the WAN uplink and downlink bandwidth is set at 3.5140 Mbps and LAN links are set at 100 Mbps, as well as in various degraded bandwidth ranging from 3.1626 (i.e., −10%) to 0.3514 Mbps (i.e., −90%), wherein 100 distributed targets (also called *marine objects*) are generated over 10 different trajectories. The scenario uses the following entities: 1 radar, 1 Automatic Identification System (AIS), Direction Finders (DFs), and 2 Electro-Optical Sensors (EOS). The oracles of the test are three: *1)* checking the number of tracks created with respect to the actual number of marine objects to detect under nominal network conditions; *2)* checking the correct detection of possible degraded network performances; *3)* check if the number of tracks created is correctly determined, even under degraded performance up to 50%. Other oracles could be considered by composing single oracles. The nominal scenario (oracle 1) is exemplified in Fig. 1, while the degraded performance scenario (oracle 2) is in Fig. 2.

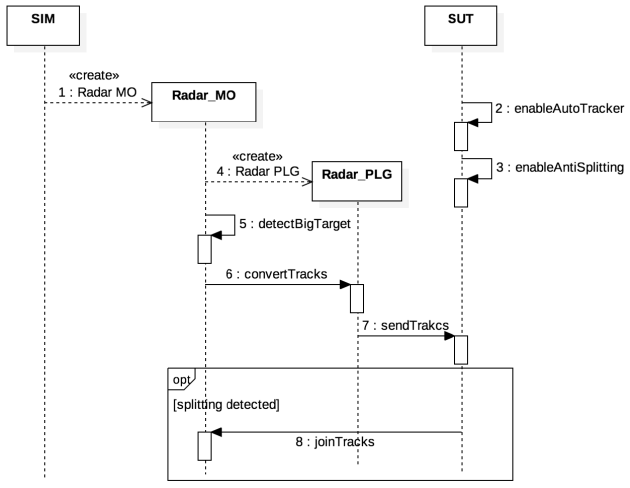

FIGURE 1. Nominal scenario.

A representative simulation setup to cope with the described scenarios like the described one is reported in Fig. 3, where the simulated VTS operation process is

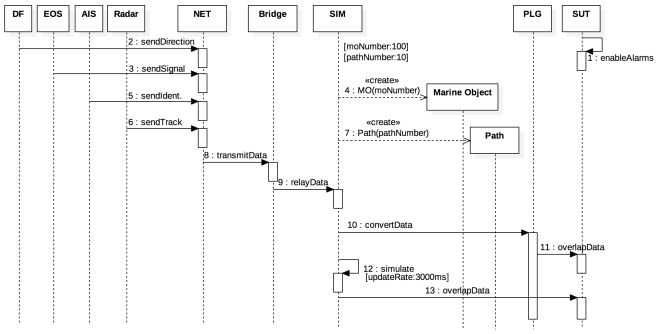

FIGURE 2. Degraded performance scenario.

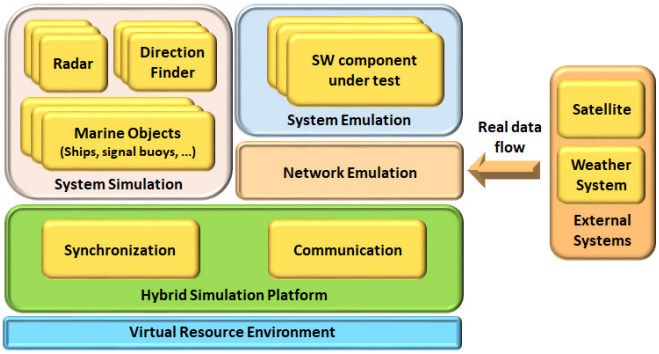

FIGURE 3. Logical view of the hybrid simulation of VTS.

obtained by the fusion of real time and simulated data, and is placed on top of a platform that supports synchronization and communication between the simulated and emulated parts for a correct evolution of the scenario.

Considering a number of desirable test scenarios, along with the mentioned considerations about the SIMPaaS approach, we have elicited a set of final-users (testers) and systems-level requirements outlined in the next paragraphs.

1) USER REQUIREMENTS

Simulation services must be accessible through Web interface, where the tester is enabled to:

- configure both the test scenarios (at architectural level) and the simulation process (dynamic behavior), to realize specific simulation experiments;
- configure the network infrastructure to interconnect distributed simulated and emulated components;
- configure virtual resources necessary to rum the simulated and emulated components;
- deploy and un-deploy a predefined simulation scenario;
- monitor the current state of the simulated scenarios.

2) SYSTEM REQUIREMENTS

User-level simulation services must be supported by system functionality. Specifically, at functional level, the simulation platform has to reproduce the test scenarios, by configuring, distributing, and deploying simulation tasks, implementing the interactions and synchronization among the involved

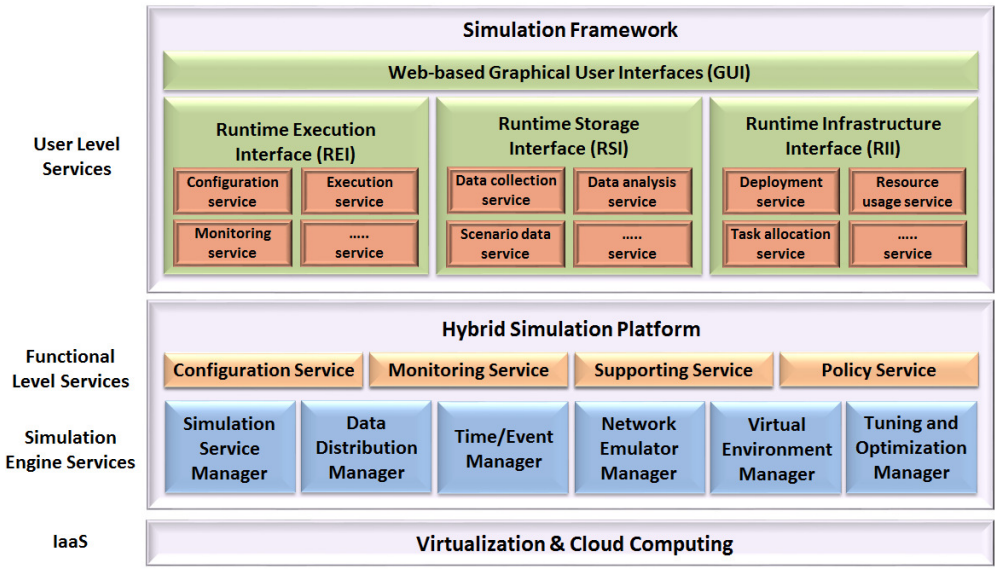

FIGURE 4. Logical view of proposed framework.

tasks, and setting up the network to interconnect simulated and emulated components; in particular:

- The complex nature of SoS forces the tester to include real/emulated entities and communication infrastructure together with simulated entities. This requires mechanisms to assure the *correct integration* of different interlinked simulation and emulation environments.
- The platform shall care about the *data distribution* between all the entities involved in the simulation, which include local and remote entities, simulated and emulated components, virtualized and real resources.
- The hybrid nature also entails complexities in the *time management*, since the simulated time of the different involved simulation tools, the physical (real) time, and the clock time should be managed. This implies the overall simulation time to be synchronized with real time of (emulated) components.
- The peculiarity of considered systems, is the strong heterogeneity among the multiple involved simulation tools. This suggests exploiting the *standards for distributed simulation*, in order to ease interoperability of different simulation/emulation environments, as well as the portability and extensibility of the framework.
- Performance and scalability are crucial. The platform should manage several independent long running simulation scenarios, which should be executed in parallel, in order to reduce testing time and cost. Thus, it should *optimize allocation of simulation tasks*, in order to reduce the resource utilization and satisfy specified confidence intervals for the results, according to tester-specified objectives.
- For the reasons explained in the previous sub-sections, the simulation should be set as a service, making most of the complexity associated with SoS transparent to the tester. This implies that the simulation platform should *manage all what concerns resource virtualization* (i.e., hardware and software configurations, communication networks, resources scheduling, scaling, etc.), orchestrating the deployment of emulated and simulated components on the virtual environment.
- Finally, while the platform can be deployed in principle on public or private cloud, when test scenarios involve critical systems and sensitive data, or due to industrial strategic reasons, simulations often demand to be performed on private clouds for *security* and *privacy* concerns.

V. HYBRID SIMULATION SERVICES

The simulation services are schematized according to four levels shown in Fig. 4: *user level*, *functional level*, *simulation engine*, and *IaaS*.

A. USER-LEVEL SERVICES

The presented framework exposes its services by means of Web-based graphical user interfaces (GUIs) and Application Programming Interfaces (APIs). In particular, specific GUIs are provided to testers in order to represent and configure test scenario, like the one presented above, by composing simulated and emulated components, and drowning their interconnections at the architecture level. The description of available components is stored in a repository. The framework will automatically represent the described scenario by an XML-based *simulation descriptors*. From logical point of view, the simulation descriptor represents the set of micro functionality involved in the simulation scenario (including individual simulation services provided by simulated/emulated components), as well as their interrelation (i.e., the structure of exchanged data) and network interconnection. In general, a scenario's descriptor includes

TABLE 1. Scenario elements and attributes.

Element/Attributes	Example or Description
Entity	
Name	*Name of the entity*
Mode	*4 Possible modes: Simulated, Emulated, Real, Plugin*
Type	*Object Type: Radar, Marine objects, Electro-Optical Sensor, Automatic Identification System, Network, ...*
Instances	*An integer number, representing the number of instances*
Parameters	*List of parameters depending on the entity Type and/or Mode,* *e.g.: Bandwidth, TrackerMode, AlarmMode, SplittingProbability, SiteMode, ...*
...	*Other attributes specific to the entity Type or Mode*
Operations	
OperationName	*Name of the entity, such as Start Radar*
Order	*Integer number giving the order of the operations sequence*
StartingEntity	*An entity of the above ones*
Action	*Type of action, e.g.: Create, Delete, Read, Write*
EndingEntity	*An entity of the above ones*
Iterations	*Number of self-calls (i.e., starting entity = ending entitiy*
Oracle	
Name	*Name of the oracle, e.g.: Check_Degradation_Uplink_Oracle*
Type	*Automatic or Manual*
Description	*Textual description of the checking to be done*
OutputEntityToCheck	*Entity to be checked, e.g.: VTX-XTH*
ExpectedOutputParam	*Parameter to be cechekd, e.g.: RadarTracks_Num*
ExpectedOutputValue	*Expected value*

the following conceptual elements, represented in Tab. 1 and described in the following:

- *Entities*, describing the objects involved in the scenario, which are distinguished in *simulated* entities, *emulated* entities, *real* entities, *plugins* (useful as adapters to connect simulated entities to the system under test, e.g., by translating different communication formats), through an attribute called *mode*. These are in turn characterized by a descriptive *type* such as: Radar, Automatic Identification System (AIS), Marine Objects, Electro-Optical Sensor (EOS), Network, and System Under Test (SUT), as well as by *parameters*, such as bandwidth, tracking mode, splitting probability, configuration parameters of AIS/EOS/Radar/. . ., etc., largely depending on the entity type and mode.
- Operations sequence, describing the sequence of operations between entities (specifying a staring and end entity) easily representable by an UML sequence diagram.
- *Oracles*, describing the expected result of a test, characterized by a type (i.e., automatic, manual check), a description, the entity to be checked against, the expected output parameter(s) and the expected output value(s) for such parameters. Complex oracles can be built by combining simple oracles.

Fig. 5 shows an example screenshot of the GUI used to deploy and monitoring a test scenario. Moreover, on the base of the specific simulation domain (e.g., VTS), specific interfaces are implemented to monitor the simulation process (for example the movement of ships in the port), as well as support the system diagnosis and log analysis.

Three sets of APIs are provided for interfacing the platform with the external environment:

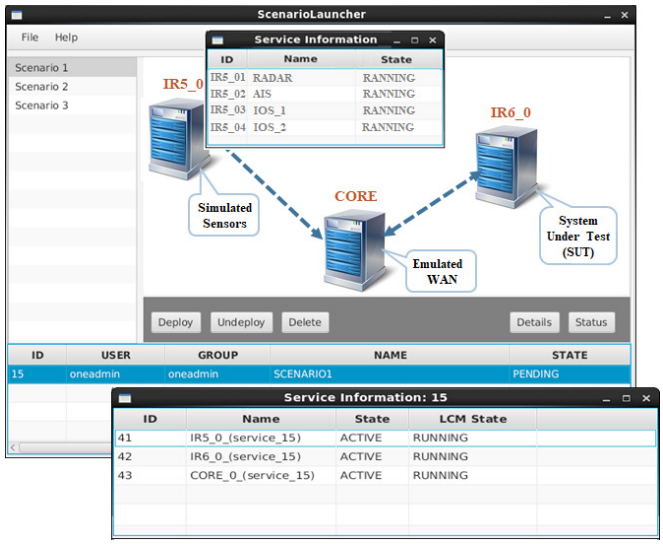

FIGURE 5. User interface of the Scenario Launcher.

- *Runtime Execution Interface* (REI) - to configure the involved simulation components and control at run time the simulation process execution.
- *Runtime Storage Interface* (RSI) - to store and manage the necessary information for orchestrating simulation, including simulated/emulated component images, configuration scenarios (simulation descriptors), simulation recording/log data, etc.
- *Runtime Infrastructure Interface* (RII) - to achieve, configure, and monitor cloud resources, according to the individual simulation process and architectural configuration, as well as deploy the simulation and emulation entities on the experimental infrastructure.

Moreover, a set of API is provided to the developer to customize the framework with respects to a specific application domain. In particular, the framework supports standard API to integrate and make interoperable existing or new simulation/emulation components in the simulation scenarios.

B. FUNCTIONAL-LEVEL SERVICES

The simulation platform uses simulation descriptors produced at user-level to deploy and execute the hybrid simulation process, which is implemented by orchestrating a combination of micro functionality offered by the involved simulated/emulated components, and macro functionality (Simulation Engine Services) exploited to implement interaction and synchronization features among them. In particular, at functional level, the following services are provided by the simulation platform:

- *Configuration and execution Service* (CS), which interacts with the lower-level modules to configure the simulation environment and instantiate simulation scenarios, providing: (*i*) the list of the simulation components and their configuration to the *Simulation Service Manager* (SSM); (*ii*) the types of data to exchange among simulated/emulated components to *Data Distribution Manager Service* (DDM); (*iii*) the setting-up of the network emulation scenario to the *Network Emulation Manager* (NEM), in order to interconnect the emulated elements with the simulated parts; and (*iv*) the architectural description of the test scenario to the *Virtual Environment Manager* (VEM), in order to implement it in the virtualized environment in a cost-effective way.
- *Monitoring Service* (MS), in which the monitoring information received back from lower modules are prompted to the tester via the GUI, in order to enable a full control of the simulation process and the state of components and the virtual environment.
- *Supporting Service*, offering repositories and services to ease the reuse of configuration scenarios, simulation/emulation components, and recording/log data.
- *Policy Service* (PS) provides services to set QoS profiles. Specifically, the user is ebabled to define policies to manage the life-cycle of components involved in the simulation. It interacts with DDM by providing the user-defined QoS profiles for data exchange among components, and with the VEM, which maps the access profiles into virtual resources access policies.

C. SIMULATION ENGINE SERVICES

This layer implements the hybrid simulation platform by means of the following modules, whose implementation is described in the next section.

- SSM – It implements the hybrid simulation process and manages the system behavior in terms of interactions among the components involved in the simulation.
- DDM – It is in charge of managing the constant exchange of information among components, providing services for data distribution.
- TEM (*Time/Event Manager*) – In order to simulate distributed and concurrent environments, it is necessary that each involved component perceives the progress of time in a uniform manner, regardless of their world of origin (real, emulated, simulated). TEM is responsible for the correct advancement of time, by managing the alignment of real and emulated components' time with the simulation time.
- NEM – Given the very complex nature of the considered network-centric systems, we adopt a distributed network emulation solution to reproduce reality with a high degree of verisimilitude. NEM implements services to create, manage and emulate complex LAN/WAN network scenarios.
- VEM – This component enables the dynamic deployment, management and monitoring of virtual resources, which are necessary for the implementation and orchestration of the local testbed. It instantiates virtual resources based on information it receives from the CS. Moreover, it is able to: (*i*) manage the life-cycle of virtual machines; (*ii*) support resource on-demand and scalability to manage multiple parallel simulation experiments; (*iii*) manage the life-cycle of the network resources and of the virtual storage.
- TOM (*Tuning and Optimization Manager*) – It implement a spatial partition algorithms for optimal scheduling and parallel execution of the simulation/emulation tasks on the virtual resources, in order to perform multiple and concurrent simulation experiments, specially when the simulation scale becomes extremely large and the resources are limited (e.g., in the case of a private cloud shared by several users and different services).

D. IaaS CLOUD SERVICES

The infrastructure layer provides the executing environment for simulation engine components and test scenarios, as well as allows fine-grained control of software stack of emulated components.

VI. SIMULATION PLATFORM AS A SERVICE

By exploiting standard for distributed simulation, virtualization and cloud paradigm, the proposed simulation platform can on-demand and elastically enable the execution of different simulation/emulated components of the VTS, and perform simulation in parallel of multiple complex and distributed test scenarios.

Specifically, in order to implement distributed simulation environment, and support simulation interoperability and synchronization, the High Level Architecture (HLA) has been adopted. It is an IEEE standard developed by the Department of Defense in the United States, which enables the reuse and interoperability of multiple independent, heterogeneous and distributed existing simulation environments, each with its own features, operating systems, and languages within a more complex federated simulation solution. By using the component-based technology, different simulation

environments can interact through standard interfaces and operate together in a federated HLA architecture composed by several interactive members, called *federate*. The interface specification of the HLA describes how to communicate within the federation, and is implemented by the Run Time Infrastructure (RTI). In order to make possible the interaction between federates and the RTI, there is the concept of ambassador, which is an interface that each federate must implement to inter-operate through the RTI middleware.

As a framework for advanced distributed interactive simulation, HLA-RTI is exploited to implement three of simulation engine services, including SSM, TEM and DDM. Specifically, as for SSM, HLA provides mechanisms for managing the federation members, as well as for specifying the interaction among the federates. Each federation member is represented by an HLA simulation object model (SOM), which specifies the types of information that a federate can provide to the federations, as well as information that it can receive from the other. The interactions among the federates are described by the federation object model (FOM), that represents the language of the federation. To correctly exchange simulation data among federates, which evolve along a different temporal model (emulated and simulated), the TEM can exploit synchronization functionality offered by RTI to coordinate how the simulators advance in their logical and emulation scenarios. According to the test scenario presented in Sec. IV, the simulated parts consist in 100 marine objects (i.e., ships, floating booms, etc.) and sensors (i.e., 1 radar, 1 AIS, and 2 EOS) used to monitor marine objects (i.e., their identification, trajectories, velocity, position, shape, etc.). The sensors will send the collected information to the SUT through the emulated network. Each a sensor will represent an HLA federate. Therefore, to share the time reference among components, specific HLA ambassadors must be implemented for each simulator integrated in the federation, which waits for events received by sensor and interacts with the others through the TEM to manage the time progress. The progress of time among distributed simulated parts is based on HLA-RTI event driven strategy [25], while the emulation parts use the system time. Specifically, during their operation, the sensors will simulate the collection of information related to the implemented scenario, and send it to the TEM. The messages received by TEM will be placed in a queue when they arrive, and are immediately eligible for delivery to the SUT by the emulated network. The ordering of these messages is arbitrary. To enable synchronization, an event driven federate will invoke the TEM, by a *Next Event Request* (NER) RTI procedure, when it has completed all simulation activity at the current logical time and is ready to advance to a new time. The parameter T specified in the NER indicates the logical time to which the federate would like to advance, if there are no other events from other federates containing a smaller time stamp. Typically, T is the time stamp of the next event in the federate's local set of pending events. If there are no messages with time stamp less than or equal to T, and none will be received in the future, the TEM invokes the federates (by *Time Advance Grant* RTI procedure) indicating its logical time has been advanced to T. Before invoking this service, the TEM will send to the SUT all federates' messages in its internal queue. Otherwise, the TEM will deliver the next smallest message destined for the federate (with time stamp T' where $T' < T$), and all other messages with time stamp T'. In this case, the federate's logical time is advanced to T'.

As regards the synchronization between the simulation and the emulation environment, under the real-time execution mode, the simulation parts follow a time-stepped discrete time model, while the network emulation uses the system time. Therefore, according to [24], assuming that the system clocks of the network emulation host and the simulation hosts are synchronized at startup time (for example, by using the network time protocol), and in the absence of delay due to communication and computation issues between the simulation and the emulation environment, such time synchronization will not violate local causality constraint. When delay exists, it will cause the time stamp discrepancy of the same message at the two environments. In the presence of delay, the simulation time should lead or equal to emulation, so that the message from simulation parts will not arrive at emulation in its past time. During our experiments, we observed that the delay varies across different experiments, with the maximum value of $35ms$ and average around $7ms$. On the other hand, VTS can be considered as a near real-time system. In a real system, the communication delay due to the WAN network can be very significant. A radar track can be considered obsoleted only it arrive with $9s$ of delay. Therefore, the delay between the two environments can be considered negligible and assimilated to a network delay.

Moreover, data exchange within the federation is implemented through the DDM, by exploiting the publish/subscribe paradigm provided by the RTI. Finally, in order to support Cloud-based distributed simulation, a Cloud-RTI middleware has been adopted, in which traditional RTI is provided by the use of Web services [26].

The NEM enhances the overall HLA-based architecture with network services needed for the interconnection of distributed simulated parts and emulated components through the emulated network over the same shared infrastructure. It interacts with the CS, from which receives the XML description of the network scenario to emulate. It can include: (*i*) the configuration of the elements to be connected; (*ii*) the type of network to emulate; (*iii*) the routing protocol; (*iv*) the network characteristics in terms of bandwidth, delay, jitter and packet loss of connections, etc. In the presented implementation the Common Open Research Emulator (CORE) has been adopted [27].

As previously described, the simulated components are objects present in the test scenario, and used to simulate the experimental workload (e.g., radars, objects in motion) or perturbation in the system operation. Each test could require several tens or hundreds of such objects to reproduce a real experimental scenario. Moreover, they could be dynamically added or removed during the the simulation,

for example, in order to simulate a failure of a remote sensor. In general, they are either software components developed from scratch or exiting simulation tools. Therefore, in order to reduce both the number of virtual nodes to be scheduled (i.e., the required computational resources, which could be limited in a private cloud), and the VTS's initialization time (boot up time) needed for the setup of the whole test scenario, we adopted two different virtualization approaches to host simulated and emulated components. Specifically, we adopt lightweight Linux containers for the simulation parts, and KVM-based full virtualization for the emulated components that must be 'physically' tested. A 'container' is a packaged self-contained, ready-to-deploy set of parts of an application. It is represented by a lightweight virtual image that can include both the middleware and business logic (binaries and libraries) to run application. Instead, a virtual machine (VM) is a full monolithic image, which requires guest OS images in addition to the binaries and libraries necessary for the applications [22]. The life cycle of containers and VMs is managed by the VEM, which provides PaaS cloud services for their packaging and deployment. VEM acts as a container manager, which enables a registry for the images of the simulated components to be deployed on the virtual nodes. It keeps track of the images executed on each node, and identifies the virtual nodes on which deploys the images, downloaded from the registry, needed for instantiating the test scenario. As Linux container, we used the Docker container virtualization technology [28]. As open-source cloud PaaS, we adopted OpenShift [29] for supporting containers, and Kubernetes [30] to orchestrate Docker containers on cluster nodes. As open-source cloud IaaS, we adopted OpenStack [33]. Moreover, an IaaS management layer has been exploited in order to access and control the virtual infrastructure, for supporting on-demand resources provisioning, running simulations on the cloud efficiently, and improving load balancing capability of the simulation. Specifically, VEM exploits Chef technology in order to simplify the configuration and deployment of the OpenShift nodes and emulated components (emulated networks and systems under test) on cloud resources [31]. In particular, software components that must be installed on virtual nodes are seen as ''resources''. ''Recipes'' specify the resources to use and the order in which they are to be applied. A ''cookbook'' defines a scenario. It contains everything that is required to support that scenario, including recipes, attribute values, file distributions, templates, etc. Chef cookbooks are stored in the Chef server (central repository), which is a component of the VEM installed on a cloud node and invoked as a remote service. The Chef Workstation loads the Chef cookbooks onto the Chef Server, and manages operations, such as the installation and execution of the agents (Chef clients) on the target nodes. Finally, the communication between the Chef workstation and the Chef server is performed by the Knife interface, which offers to the testers management functionality for nodes, recipes and cookbooks, roles.

Finally, a resource management algorithm has been proposed in our previous work, which optimizes allocation of the simulation tasks (i.e., OpenShift nodes) and emulated components on virtual nodes, reducing the cost to the service provider (the cloud resource consumption in a private cloud), and enhances the parallelization of the simulation jobs, by fanning out more federated instances (test scenarios) [34]. The computed allocation of the test scenario is translated in a Chef cookbooks and automatically deployed by VEM.

VII. TEST SCENARIOS

The developed platform is used to implement several test scenarios of the VTS of the DISPLAY project's industrial partner. In the following, we describe the implemented testbed and the test scenarios that can be currently run in the hybrid simulation mode without the burden (or infeasibility, in various cases) of a real implementation.

A. TESTBED

Figure 6 schematizes the implemented testbed.

It is based on a DELL M820 blade cluster with 32 nodes, each with two quad-core processors, a 72GB HDD and either 8/16/32 GB RAM. Each node is equipped with the *CentOS 6.6 x86_64* distribution – kernel *2.6.32-504.8.1.el6x86_64*, *OpenSwitch* (OVS) 2.3.1 linked as kernel module, and Openstack Icehouse series. Each node is equipped with 4 network interface cards (NIC) at 1Gbps. One NIC is used by the DELL management suite; two NICs are used for the data exchange among virtualized nodes; the fourth NIC is used by *OpenNebula* for node and VMs monitoring.

The logical components running on the testbed are:

- The System Under Test (SUT), namely the VTS, whose behaviour is observed under the given test scenario;
- The simulation platform (SIM), responsible for simulating data coming from many types of sensors, such as Radars, AISs, EOSs, DFs, Weather station and others;
- The plugin component (PLG), which is an adapter to convert data format from/to SIM to/from the SUT; implemented as a HLA federate; moreover, a *Bridge* that do not interact with the SIM, relays data captured from the operational site toward the SUT;
- The network component (NET), providing the network infrastructure, that is interfaced with external systems by the routers emulated bu CORE; moreover, a DMZ is used to decouple and protect the interface from the OS and from the DB, for security reasons.

The high-level objective of the testbed is to provide the ability to setup realistic scenarios with a desired and configurable (at runtime) number of marine objects (e.g., ships, buoys, wrecks) along with simulated data coming from all kinds of sensors (e.g., radars, AISs, EOSs, DFs) from a given geographical site, provided as input to the VTS under test.

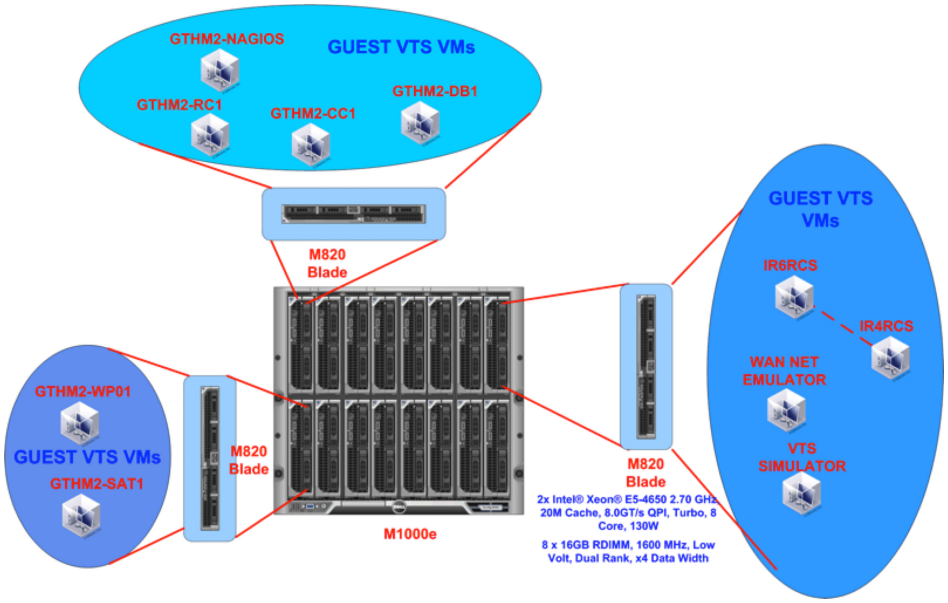

FIGURE 6. Testbed.

B. TEST SCENARIOS

By using the platform, we implemented all the operational scenarios reported in Tables 2–5. They are grouped into the following four types:

- *Maritime traffic control operation (MTC)* - Scenarios in this category test the correct functioning of the VTS instance (emulated on the developed platform) with respect to maritime traffic control operations (i.e., they mimic real operational scenarios, according to a given functional test plan drawn from requirements).
- *Network operation (N)* - These scenarios aim at recreating WAN contexts, to test the correct functioning of the VTS with respect to varying network conditions.
- *Fault tolerance (FT)* - These scenarios are reproduced to test conditions where faults (at component and/or network level) can occur, so as to check to detection and recovery ability of the VTS.
- *Multiple scenarios (M)* - Tests in this category aim at checking the behaviour of the VTS when multiple instances are present in one center and/or when it operates in multiple control centers at the same time.

The scenarios listed in Table 2–5 act as *test frames*, namely as a suitable combination of classes of parameters' values from which many concrete test cases can be drawn (by instantiating specific values from within the classes range). Each scenario is characterized by: an ID denoting the group it belongs to (among the four ones mentioned above); the high-level objective of the scenario; the entities involved (simulated, emulated or real entities, depending on the scenario); the relevant parameters along with the ranges of input that can be provided.[1] From these frames, testers easily obtain test case by specifying values for the entities and for the relevant parameters,

As a matter of fact, the new platform practically enabled a pre-release system testing activity (both functional and non-functional) that was previously prohibitive: without the platform, many of those scenarios were unfeasible (or extremely expensive), and thus, single pieces tested in isolation, because of many real systems involved in geographically distributed sites and not synchronously available for testing. The platform enables the possibility of running tests continuously, as the platform is system-independent. Also, it allows a drastic reduction of orders of magnitude (from days to few hours or minutes) to setup and run a single test case with respect to a (feasible) real test. The 19 scenarios we implemented are expected to increase with the increase of the platform usage by the industrial partner's engineers. This will expectedly bring to considerable saving of testing time/cost and many more scenarios that can be actually proved.

VIII. RELATED WORK

Simulation Software-as-a-Service (SIMSaaS) is a relatively new paradigm that has achieved significant attention on in the Cloud Computing community, which enables the execution of simulation application environments that can be deployed on-demand and offered as-a-service [35]. Several specific application domains use SIMSaaS, such as traffic and transportation [36], scheduling parallel discrete event simulation jobs [37], and manufacturing [38]. Tsai *et al.* [19] proposed a generic SIMSaaS framework incorporating multi-tenancy architecture and scalability for simulation, also presenting a simulation runtime infrastructure. The Fortissimo project provides one-stop, pay-per-use, on-demand access to simulation cloud resources, including software, hardware and expertise [41]. In particular, it focuses on modeling and simulation of coupled physical processes and high-performance

[1]Ranges' boundary values are determined by the tester, who usually observes typical runtime conditions to set them, and/or refers to extreme conditions, depending on the objective of the test.

TABLE 2. Maritime traffic control test scenarios. Legend: AIS = Automatic Identification System; EOS = Electro-Optical Sensor; DF = Direction Finder; MO = Marine Objects; NM = Nautical Mile; SAR = Synthetic Aperture radar.

ID	Objective	Entities	Relevant Parameters	Input (Range)
MTC_1	Check Performance (response time)	1 AIS Radars EOSs DFs MOs	- #Radars #EOSs #DFs #MOs #Paths Update period WAN Bandwidth LAN Bandwidth	- 5 to 20 5 to 20 5 to 20 100 to 500 20 to 100 3000ms 100 Mbps 3,5140 Mbps
MTC_2	Check the Tracker resolution	1 Radar MOs	- #MOs Distance among MOs	- 2 to 5 1m to 400m
MTC_3	Check the behaviour of the Tracker upon tracks fusion and under bandwidth reduction	1 Radar 1 AIS MOs	- - #MOs Bandwidth Reduction	- - 2 to 5 10% to 50%
MTC_4	Check the antisplitting ability of the Tracker	1 Radar 1 MO	- MO Splitting Prob.	- 0 to 1
MTC_5	Check the tolerance to a faulty transponder by verifying the creation of equal tracks under both correct and wrong information by transponder (simulated by a track jump)	1 AIS 1 MO	- - Track Jump	- - 0.1 NM to 2 NM
MTC_6	Check the correct computation of collision times given by the Distance Time Diagram (DTD)	1 AIS 1 Radar MOs	- - #MOs	- - 10 to 50
MTC_7	Check the correct activation of a *collision alarm* under at minimal distance threshold	1 AIS 1 Radar MOs	- - #MOs Distance Threshold	- - 10 to 50 1 NM to 5 NM
MTC_8	Check the correct notification of a *safety message* received by a MO	1 AIS 1 MO	- -	- -
MTC_9	Check the correct positioning of targets on a map referring to a very large area	1 AIS 1 Radar 1 SAR MOs	- - - #MOs Avg MO Distance	- - - 10 to 50 100 NM to 500 NM

TABLE 3. Network test scenarios. Legend: AIS = Automatic Identification System; EOS = Electro-Optical Sensor; MO = Marine Objects.

ID	Objective	Entities	Relevant Parameters	Input (Range)
N_1	Check the tolerance to the WAN bandwidth reduction up to a threshold (i.e., check the correct creation of tracks)	1 AIS 1 Radar 2 EOSs MOs	- - - #MOs #Paths WAN Band Reduction	- - - 100 to 500 2 to 10 -10% to 90%
N_2	Check the tolerance to faults (downtime) in the WAN link between a remote site and the Control Center (i.e., the maximum uptime under network failure and the time to notify the failure are as expected)	3 AISs 4 Radars MOs 1 Remote Site	- - #MOs - #Paths WAN Downtime	- - 100 to 500 - 2 to 10 10s to 180s
N_3	Check the tolerance to faults (downtime) in the WAN link among multiple Control Centers (i.e., the maximum uptime under network failure and the time to notify the failure are as expected)	3 AISs 4 Radars MOs 1 Remote Site 2 Control Centers	- - #MOs - - #Paths WAN Downtime	- - 100 to 500 - - 2 to 10 10s to 180s
N_4	Check the tolerance to latency in the WAN link up to a threshold (i.e., check the correct creation of tracks)	1 AISs 1 Radar 2 EOSs	- - - #Paths Latency	- - - 2 to 10 $\pm$50ms to $\pm$500ms

data analytic by exploiting HPC facilities. However, there is a lack of automation and integration of tools in modeling and simulation of distributed systems.

In this direction, Distributed Interactive Simulation (DIS) and HLA represent two standards for distributed simulation. DDS is managed by the Object Management Group, and

TABLE 4. Fault tolerance test scenarios. Legend: AIS = Automatic Identification System; EOS = Electro-Optical Sensor; MO = Marine Objects; SAR = Synthetic Aperture radar.

ID	Objective	Entities	Relevant Parameters	Input (Range)
FT_1	Check the tolerance and recovery ability of the SUT under a failure of a node	3 AIS 4 Radar 1 SAR 2 SUTs MOs	- - - - #MOs #Paths	- - - - 100 to 500 2 to 10
FT_2	Check the tolerance and recovery ability of the SUT under the failure of a LAN link	3 AIS 4 Radar 1 SAR 2 SUTs MOs	- - - - #MOs #Paths	- - - - 100 to 500 2 to 10
FT_3	Check the tolerance and recovery ability of the SUT under the failure of multiple nodes and LAN links	3 AIS 4 Radar 1 SAR 2 SUTs MOs	- - - - #MOs #Paths	- - - - 100 to 500 2 to 10
FT_4	Check the tolerance and recovery ability of the SUT under the failure of one or more switches	3 AIS 4 Radar 1 SAR 2 SUTs MOs	- - - - #MOs #Paths	- - - - 100 to 500 2 to 10

TABLE 5. Multiple test scenarios. Legend: AIS = Automatic Identification System; EOS = Electro-Optical Sensor; DF = Direction Finder; MO = Marine Objects; NM = Nautical Mile; SAR = Synthetic Aperture radar.

ID	Objective	Entities	Relevant Parameters	Input (Range)
M_1	Check the correct positioning of targets with multiple instances of the SUT in one center	1 AIS 1 Radar 1 SAR 2 SUTs MOs SUT 1 MOs SUT 2	- - - - #MOs #MOs #Paths SUT 1 #Paths SUT 2	- - - - 10 to 50 10 to 50 2 to 10 2 to 10
M_2	Check the correct positioning of targets with one SUT on multiple centers	1 AIS 1 Radar 1 SAR 2 SUTs MOs	- - - - #MOs #Paths	- - - - 10 to 50 2 to 10

used as a messaging middleware standard for supporting data-centric simulations, enabling seamless, timely, scalable, and dependable distributed data sharing [39]. However, it focuses exclusively on information exchange to support the federation of solutions without providing the necessary introspection [40]. Its successor HLA is a bit more flexible, as the information to be exchanged is not standardized (it only says how to structure the data). HLA is essentially used to facilitate the reuse and interoperability of different simulation systems and assets. On the other hand, HLA does not well suit the considered large-scale, fine-granularity and long-time distributed simulation scenarios, for its inefficient utilization of simulation resources, lack of load balancing capability, weak fault tolerance capability, and complicated simulation deployment process [26]. RTI services are centralized, which can be a bottleneck of the interaction between large-number of hosted simulation entities. Therefore, the new concept that includes service orientation and provision of simulation applications via the PaaS model of Cloud Computing would allow overcoming the discussed limits.

One of the first cloud-based HLA approach is introduced in [42]. It presents a possible solution for integrating HLA with a Service Oriented Architecture (SOA) in the context of a smart building project. The simulation manager module is a service wrapper on top of the RTI, which exposes access to the RTI federation services via a RESTful API. RESTful-based RTI have been used in the present work to make interoperable the simulated VTS components hosting on distributed cloud virtual nodes.

HLAcloud [43] presents a model driven and cloud-based framework to support both the implementation of distributed HLA-based simulation systems from a SysML (Systems Modeling Language) specification of the system under study, and its execution over a cloud infrastructure. It generates

the Java/HLA source code of the federates and the scripts required to deploy and execute the HLA federation onto the PlanetLab cloud-based infrastructure. Similarly to such contribution, this work aims to propose a framework to provide distributed simulation on Cloud infrastructure. Nevertheless, in this work a simulation PaaS has been implemented to make interoperable simulated and emulated distributed components of a VTS, as well as automatically configure deploy lightweight Linux containers for the simulation parts and KVM-based virtual nodes for the emulated components on open-source cloud IaaS, such as OpenStack.

IX. CONCLUSIONS AND FUTURE WORK

Hybrid and distributed simulation, supported by novel technologies for resources virtualization and working environment reproduction, represents the most promising way to define the strategies needed to actually support SoS testing in factory as it would be on-site. Designing an architecture for scalable, distributed and parallel simulation, with automated resource management and execution on the cloud, can satisfy the requirements of large-scale simulations effectively.

The proposed solution provides testers with a distributed simulation platform able to support the implementation of *in factory* local testbeds for reproducing complex *in situ* test scenarios. The framework is able to integrate heterogenous emulated and simulated environments by relying on synchronization, communication and virtualization services and is itself offered ''as a service'' to testers, who can define arbitrarily complex scenarios without caring about the underlying integration and communication complexity. The framework is currently adopted by VTS engineers for building large-scale, realistic and effective test scenarios with remarkable gains in terms of tests set up and execution cost.

REFERENCES

[1] J. Guckenheimer and J. M. Ottino, ''Foundations for complex systems research in the physical sciences and engineering,'' Nat. Sci. Found., Tech. Rep., 2008. [Online]. Available: http://pi.math.cornell.edu/~gucken/PDF/nsf_complex_systems.pdf

[2] P. Pederson, D. Dudenhoeffer, S. Hartley, and M. Permann, ''Critical infrastructure interdependency modeling: A survey of US and international research,'' Idaho Nat. Lab., Idaho Falls, ID, USA, Tech. Rep. INL/EXT-06-11464, 2006.

[3] M. Ouyang, ''Review on modeling and simulation of interdependent critical infrastructure systems,'' *Rel. Eng. Syst. Saf.*, vol. 121, pp. 43–60, Jan. 2014.

[4] *DISPLAY—Distributed Hybrid Simulation Platform for ATM and VTS Systems*. Accessed: Aug. 2018. [Online]. Available: http://www.dieti.unina.it/index.php?option=com_content&view=article&id=255:display&catid=75&Itemid=341&lang=it

[5] A. Souid, A. Delaplace, F. Ragueneau, and R. Desmorat, ''Pseudodynamic testing and nonlinear substructuring of damaging structures under earthquake loading,'' *Eng. Struct.*, vol. 31, no. 5, pp. 1102–1110, 2009.

[6] D. De Laurentis, ''Role of humans in complexity of a system-of-systems,'' in *Digital Human Modeling* (Lecture Notes in Computer Science), vol. 4561. Berlin, Germany: Springer, 2007, pp. 363–371.

[7] IEEE-Reliability Society. (2014). *Technical Committee on 'Systems of Systems'-WHITE PAPER*. [Online]. Available: http://rs.ieee.org/component/content/article/9/77-system-of-systems.html

[8] T. Aven, ''Interpretations of alternative uncertainty representations in a reliability and risk analysis context,'' *Rel. Eng. Syst. Saf.*, vol. 96, no. 3, pp. 353–360, 2011.

[9] E. Zio, *Computational Methods for Reliability and Risk Analysis*. Singapore: World Scientific, 2009.

[10] L. Dueñas-Osorio, I. C. James, J. G. Barry, and B. Ann, ''Interdependent response of networked systems,'' *J. Infrastruct. Syst.*, vol. 13, no. 3, pp. 185–194, 2007.

[11] J. Johansson and H. Hassel, ''An approach for modeling interdependent infrastructures in the context of vulnerability analysis,'' *Rel. Eng. Syst. Saf.*, vol. 95, no. 12, pp. 1335–1344, 2010.

[12] *Mosaik—A Flexible Smart Grid Co-Simulation Framework*. Accessed: Aug. 2018. [Online]. Available: https://mosaik.offis.de/

[13] *RinSim—A Simulator for Logistics Problems*. Accessed: Aug. 2018. [Online]. Available: https://github.com/rinde/RinSim

[14] *COSSIM—A Novel, Comprehensible, Ultra-Fast, Security-Aware CPS Simulator*. Accessed: Aug. 2018. [Online]. Available: http://www.cossim.org

[15] *MARIN—Vessel Traffic Service (VTS) Simulator*. Accessed: Aug. 2018. [Online]. Available: http://www.marin.nl/web/Facilities-Tools/Simulators/Simulator-Facilities/VTS-Simulators.htm

[16] *K-Sim VTS—Vessel Traffic Services Simulator*. Accessed: Aug. 2018. [Online]. Available: https://kongsberg.com/en/kongsberg-digital/maritime%20simulation/vessel%20traffic%20services%20simulator/

[17] L. Braubach and A. Pokahr, ''The Jadex project: Simulation,'' in *Multiagent Systems and Applications*, vol. 45. Berlin, Germany: Springer, 2013, pp. 107–128.

[18] S. Shekhar, H. Abdel-Aziz, M. Walker, F. Caglar, A. Gokhale, and X. Koutsoukos, ''A simulation as a service cloud middleware,'' *Ann. Telecommun.*, vol. 71, no. 3, pp. 93–108, 2016.

[19] W.-T. Tsai, W. Li, H. Sarjoughian, and Q. Shao, ''SimSaaS: Simulation software-as-a-service,'' in *Proc. 44th Annu. Simulation Symp. (ANSS)*, 2011, pp. 77–86.

[20] S. Guo, F. Bai, and X. Hu, ''Simulation software as a service and service-oriented simulation experiment,'' in *Proc. IEEE Int. Conf. Inf. Reuse Integr.*, Aug. 2011, pp. 113–116.

[21] T. Preisler, T. Dethlefs, and W. Renz, ''Simulation as a service: A design approach for large-scale energy network simulations,'' in *Proc. Federated Conf. Comput. Sci. Inf. Syst.*, vol. 5, Sep. 2015, pp. 1765–1772.

[22] C. Pahl, ''Containerization and the PaaS cloud,'' *IEEE Cloud Comput.*, vol. 2, no. 3, pp. 24–31, May/Jun. 2015.

[23] M. Ficco, G. Avolio, F. Palmieri, and A. Castiglione, ''An HLA-based framework for simulation of large-scale critical systems,'' *Concurrency Comput., Pract. Exper.*, vol. 28, no. 2, pp. 400–419, 2016.

[24] J. Sztipanovits, ''A model-based integration of network emulation with HLA-based heterogeneous simulation environments,'' Inst. Softw. Integr. Syst., Nashville, TN, USA, Tech. Rep. (ISIS)-10-107, 2010.

[25] R. M. Fujimoto, ''Time management in the high level architecture,'' *Simulation*, vol. 71, no. 6, pp. 388–400, 1998.

[26] H. Heng, R. Li, X. Dong, Z. Zhang, and H. Han, ''An efficient and secure cloud-based distributed simulation system,'' *J. Appl. Math. Inf. Sci.*, vol. 6, no. 3, pp. 729–736, 2012.

[27] J. Ahrenholz, ''Comparison of CORE network emulation platforms,'' in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Oct. 2010, pp. 864–869.

[28] D. Merkel, ''Docker: lightweight Linux containers for consistent development and deployment,'' *Linux J.*, vol. 2014, no. 239, Mar. 2014, Art. no. 2. Accessed: Aug. 2018. [Online]. Available: http://dl.acm.org/citation.cfm?id=2600239.2600241

[29] *OPENSHIFT—Red Hat Openshift*. Accessed: Aug. 2018. [Online]. Available: https://www.openshift.com/

[30] *Kubernetes is an Open-Source System for Automating Deployment, Scaling, and Management of Containerized Applications*. Accessed: Aug. 2018. [Online]. Available: http://kubernetes.io/

[31] *CHEF—Cloud Management*. Accessed: Aug. 2018. [Online]. Available: https://www.chef.io/solutions/cloud-management/

[32] *Using Chef to Customize Multi-Node Cloud Foundry Deployments*. Accessed: Aug. 2018. [Online]. Available: https://blog.pivotal.io/pivotal-cloud-foundry/products/using-chef-to-customize-multi-node-cloud-foundry-deployments

[33] *OpenStack—Open Source Software for Creating Private and Public Clouds*. Accessed: Aug. 2018. [Online]. Available: https://www.openstack.org/

[34] M. Ficco, B. Di Martino, R. Pietrantuono, and S. Russo, ''Optimized task allocation on private cloud for hybrid simulation of large-scale critical systems,'' *Future Gener. Comput. Syst.*, vol. 74, pp. 104–118, Sep. 2017.

[35] T. Azevedo, J. F. Rosaldo Rossetti, and G. Jorge Barbosa, "A state-of-the-art integrated transportation simulation platform," in *Proc. 4th Int. Conf. Models Technol. Intell. Transp. Syst.*, Jun. 2015, pp. 340–347.
[36] J. Harri, M. Killat, T. Tielert, J. Mittag, and H. B. Hartenstein, "DEMO: Simulation-as-a-service for its applications," in *Proc. 71st IEEE Veh. Technol. Conf. (VTC)*, May 2010, pp. 1–2.
[37] X. Liu, X. Qiu, B. Chen, Q. He, and K. Huang, "Scheduling parallel discrete event simulation jobs in the cloud," in *Proc. IET Conf. Publications*, 2012, p. 72.
[38] S. J. E. Taylor, T. Kiss, G. Terstyanszky, P. Kacsuk, and N. Fantini, "Cloud computing for simulation in manufacturing and engineering: Introducing the CloudSME simulation platform," in *Proc. Annu. Simulation Symp.*, 2014, pp. 89–96.
[39] *IEEE Standard for Distributed Interactive Simulation—Application Protocols*, IEEE Standard 1278.1, 2012.
[40] A. Tolk and S. Y. Diallo, "Using a formal approach to simulation interoperability to specify languages for ambassador agents," in *Proc. Winter Simulation Conf. (WSC)*, 2010, pp. 359–370.
[41] *The Fortissimo Project*. Accessed: Aug. 2018. [Online]. Available: https://www.fortissimo-project.eu
[42] M. Dragoicea, L. Bucur, W.-T. Tsai, and H. Sarjoughian, "Integrating HLA and service-oriented architecture in a simulation framework," in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCgrid)*, May 2012, pp. 861–866.
[43] P. Bocciarelli, A. D'Ambrogio, A. Giglio, and D. Gianni, "A SAAS-based automated framework to build and execute distributed simulations from SysML models," in *Proc. Winter Simulation Conf.*, Dec. 2013, pp. 1371–1382.

MASSIMO FICCO (M'02) received the degree in informatics engineering from the Università degli Studi di Napoli Federico II in 2000 and the Ph.D. degree in information engineering from the Parthenope University of Naples in 2010. From 2000 to 2010, he was a Senior Researcher with the Italian University Consortium for Computer Science. Since 2004, he has been teaching master courses in software reliability and security, software engineering, data bases, and programming. He is currently an Assistant Professor with the Università degli Studi della Campania Luigi Vanvitelli. His current research interests include security and reliability of critical infrastructure, cloud computing, and mobile computing.

ROBERTO PIETRANTUONO (SM'16) is currently an Assistant Professor with the Università degli Studi di Napoli Federico II, where he teaches software engineering. He has co-authored over 60 papers in his research areas. His research interests are in the areas of software reliability engineering, software testing, and verification of critical software systems. He is a Co-Founder of Critiware s.r.l., a company working in critical systems engineering. Since 2008, he has been involved in several EU and national projects on software engineering and software dependability.

STEFANO RUSSO (SM'15) has been a Professor of computer engineering with the Università degli Studi di Napoli Federico II since 2002, where he teaches software engineering and distributed systems and leads the Dependable Systems and Software Engineering Research Team. He has co-authored over 160 papers in the areas of software engineering, software aging, middleware technologies, and mobile computing. He is an Associate Editor of the IEEE TRANSACTIONS ON SERVICES COMPUTING.