A MICROSERVICE-BASED APPROACH FOR FINE-GRAINED SIMULATION IN MSAAS PLATFORMS

Paolo Bocciarelli
Andrea D'Ambrogio
Andrea Giglio
Emiliano Paglia

Department of Enterprise Engineering
University of Rome Tor Vergata
Via del Politecnico, 1 Rome, Italy
{paolo.bocciarelli,dambro,andrea.giglio,emiliano.paglia}@uniroma2.it

ABSTRACT

M&S as a Service (MSaaS) is an increasingly adopted paradigm that brings the benefits of service-oriented architectures and cloud computing into the M&S field. The design and implementation of MSaaS platforms typically address the provision of coarse-grained M&S services, which offer the user easy access and orchestration of M&S components consisting of entire environments, applications and/or tools. This paper introduces an approach to the provision of fine-grained M&S services, which are defined by use of a microservice-based architecture, according to which applications are developed as a suite of small-sized services. The proposed approach extends an already available MSaaS platform, named SOASim. The paper shows how the integration and mutual use of fine-grained and coarse-grained services (e.g., modeling services, transformation services, presentation services etc.) significantly enhance the benefits of SOASim. An example application to the microservice-based setup of a discrete event simulation is used to describe and discuss the proposed approach.

Keywords: Microservices, DES, Cloud, SaaS, MSaaS.

1 INTRODUCTION

Modeling & Simulation (M&S) approaches are widely recognized as effective and valuable solutions that allow decision-makers and system engineers to get the appropriate understanding of complex systems and processes, at various levels of abstraction and lifecycle stages. (Gianni et al. 2014).

In short, a conventional M&S approach is carried out by first building a simulation model that represents a system or process under study at a given level of abstraction, and then transforming the simulation model into an executable software implementation, which is eventually run to reproduce the behavior of the system or process according to its intended use and addressed conditions (Sokolowski and Banks 2009).

Despite its acknowledged relevance, the actual adoption of M&S approaches is still limited, mostly due to the significant know-how, effort and cost that are usually needed to build and implement simulation models, as well as to deploy and execute the corresponding model implementations, especially in case of distributed execution platforms.

In order to overcome the aforementioned limitations and better exploit the full potential of M&S, a new paradigm has recently emerged from the introduction of service orientation and cloud computing in the M&S field. Such a paradigm, denoted as *M&S-as-a-Service (MSaaS)*, is intended to improve the reusability and cost-effectiveness of M&S efforts, as well as to address significant properties in terms of accessibility, composability and interoperability (Cayirci 2013, Siegfried et al. 2018).

Specifically, service orientation has proven to be a valuable approach for building distributed software systems throughout the orchestration of loosely coupled software services (Papazoglou and Georgakopoulos 2003), whereas the adoption of cloud computing is recognized as a cost-effective solution to deploy and execute services that provide access to resources of various types, such as infrastructures, platforms and software (Hayes 2008).

The design and implementation of MSaaS platforms typically address the provision of *coarse-grained M&S services*, which offer the user easy access and orchestration of M&S resources consisting of entire applications, environments and/or tools. Example MSaaS platforms providing coarse-grained M&S services are CloudSME (Taylor et al. 2018), the Simulation Platform (Yamazaki et al. 2011) and the OCEAN platform (Biagini et al. 2017).

This paper introduces an approach to the provision of *fine-grained M&S services*, which are designed by use of a *microservice-based architecture*, according to which applications are developed as a suite of small-sized services, each running in its own process and communicating with lightweight mechanisms (Daya et al. 2016).

An example MSaaS platform providing coarse-grained M&S services is the DEVS-MF platform, which allows users to easily build DEVS-based discrete event simulation systems by orchestrating resources consisting of DEVS atomic and/or coupled simulators. This paper contribution introduces an approach of finer granularity and formalism independent, thus opening the way to the interoperability among simulators addressing different formalisms (e.g., DEVS and non-DEVS simulators).

In addition, the proposed approach is integrated into an already existing MSaaS platform, named *SOASim*, which includes a set of coarse-grained M&S services for simulation modeling, model transformation, automated model implementation and output visualization (D'Ambrogio et al. 2016). The paper shows how the integration and mutual use of newly introduced fine-grained services and already existing coarse-grained services (e.g., modeling services, transformation services, visualization services, etc.) significantly enhance the benefits of SOASim.

This paper contribution is illustrated through an example application dealing with the microservice-based setup of a queueing-based *discrete event simulation (DES)*.

The rest of the paper is structured as follows. Section 2 briefly overviews relevant concepts and technologies at the basis of this work. Section 3 outlines the SOASim platform and its extension for the provision of fine-grained M&S services. Section 4 illustrates how a given simulation can be developed, deployed and executed by use of SOASim, as further described in Section 5 by use of an example application to a conventional DES. Finally, Section 6 discusses and summarizes the proposed approach.

2 BACKGROUND

This section provides the necessary background about the main concepts addressed by this work, namely microservices, cloud computing and containerization.

2.1 Microservices

Microservices are small, autonomous services that interact using a standard communication protocol and a set of well-defined APIs (Application Programming Interfaces) and events, independent of any vendor, product or technology (Karmel et al. 2016, Newman 2015).

According to (Daya et al. 2016), microservices are:

- small and focused: microservices need to focus on a well-defined unit of work, and as such they should be small enough to be managed by a single team;
- loosely coupled: a microservice must be designed by reducing dependencies on other microservices;
- language-neutral: even though microservices are integrated to build complex applications, they do not need to use the same programming language or any other common underlying technology;
- data decentralized: ideally, a microservice should store data in its own database. A complex application, built by integrating several different services, should not deal with transactions that are spread across multiple services;
- discoverable: microservices should be located and identified by use of discovery services.

According to such a characterization, the design of a microservice architecture must address two different issues: distributed data management, which has to be enacted in order to store a microservice state in local databases, and shared event storage and processing, to facilitate the information exchange between stateless microservices. Information from local databases is thus used in conjunction with the event processing inside the microservice in order to execute the business logic. The fundamental benefits of microservices can be summarized as follows:

- improved scalability: services can scale easily and independently from other services;
- improved agility: thanks to the high degree of modularity, a change to a service has a minimal impact on any other service;
- improved application availability: if a partial service failure occurs, it is highly unlikely that it will directly impact the rest of the application;
- continuous delivery: changes or updates on a particular service can be carried out without setting up the entire application in maintenance mode.

2.2 Cloud Computing: Delivery Paradigms and Containerization

Cloud computing provides ubiquitous and on-demand network access to computing resources (e.g., networks, servers, storage, applications, and services) delivered as a service (Mell and Grance 2011). The pool of resources can be easily provisioned and configured in a scalable and dynamic way, to gain a reduction of the costs associated with the management of complex computational infrastructures (Vaquero, Rodero-Merino, Caceres, and Lindner 2008, Hayes 2008).

Cloud computing is founded on an everything-as-a-service delivery model, which enables the provision of applications (Software-as-a-Service, SaaS), development platforms (Platform-as-a- Service, PaaS) and computing infrastructures in terms of storing and processing capacity (Infrastructure-as-a-Service, IaaS) (Menascé and Ngo 2009).

A basic building block of the cloud computing paradigm is the concept of *containerization*. A container is a self-contained runtime environment for a software application, which can be used to wrap services in order to ease their deployment and execution in the cloud. Containers provide a means to virtualize an operating systems so that multiple workloads can run on a single operating system instance, differently from *virtual machines*, which virtualize the hardware to run multiple operating system instances (Karmel, Chandramouli, and Iorga 2016). Specifically, the proposed MSaaS platform adopts the *Docker* containerization technology (Docker 2016). A Docker container wraps a piece of software in a complete system that contains everything needed to run the software: code, runtime, system tools, system libraries and anything that can be installed on a server, guaranteeing that the software will always run the same, regardless of its hosting environment.

3 SOASIM PLATFORM OVERVIEW

The SOASim platform has been proposed to provide a service-oriented framework to effectively support the building, deployment and execution of simulation experiments in an MSaaS paradigm (D'Ambrogio et al. 2016, Bocciarelli et al. 2017). Under a general perspective, SOASim provides services which address the steps required to execute a simulation-based system analysis, namely *modeling services*, to specify the abstract model of the system under study, *transformation services*, to generate the corresponding executable simulation model, *simulation services*, to execute the model implementation, and *presentation services*, to summarize and visualize the analysis results.

SOASim has been initially developed to provide coarse-grained M&S services, such as modeling services to build abstract models of systems or simulation models in a given language, and simulation services to execute monolithic simulation models. This work addresses instead a different level of abstraction and introduces a fine-grained approach to the provision of M&S services, which have been eventually integrated in the SOASim platform.

In general, a simulation model can be seen as a composition of various instances of the simulation language primitives (e.g., queues and service centers for a queueing network model, tasks and resources for a business process model, places and transitions for a petri net model), as well as additional components that are required to orchestrate and execute the simulation model (e.g., schedulers, event calendars and clocks, for a conventional discrete event simulation). The basic idea of this paper contribution is to provide a set of microservices, each one implementing a single small and highly reusable simulation primitive or component. In addition, the integration of such microservices into the SOASim platform allows users to take advantage of already available coarse-grained services, such as modeling services, data analysis and presentation services and discovery services.

The architecture of the SOASim platform is shown in Figure 1. The bottom layer, named *Cloud Infrastructure Layer*, provides two different services:

- **Storage Service**: the cloud-based repository which stores the several data and entities used by the M&S services: *Models*, which can be generated by a model transformation or directly specified by use of a modeling service, *Data* and *Microservices images*, which constitute the core of the SOASim plaftorm.
- **Computation Service**: the set of computation nodes made available by use of Docker virtual containers, in which service images are deployed and executed.

The outermost layer of SOASim is the *Web Layer*, which implements the user interface by use of a web portal. The *Middleware Layer* provides access to the M&S services available on the underlying cloud infrastructure, by use of the following components:

- **User Identification and Authorization**: component that provides the authentication services.

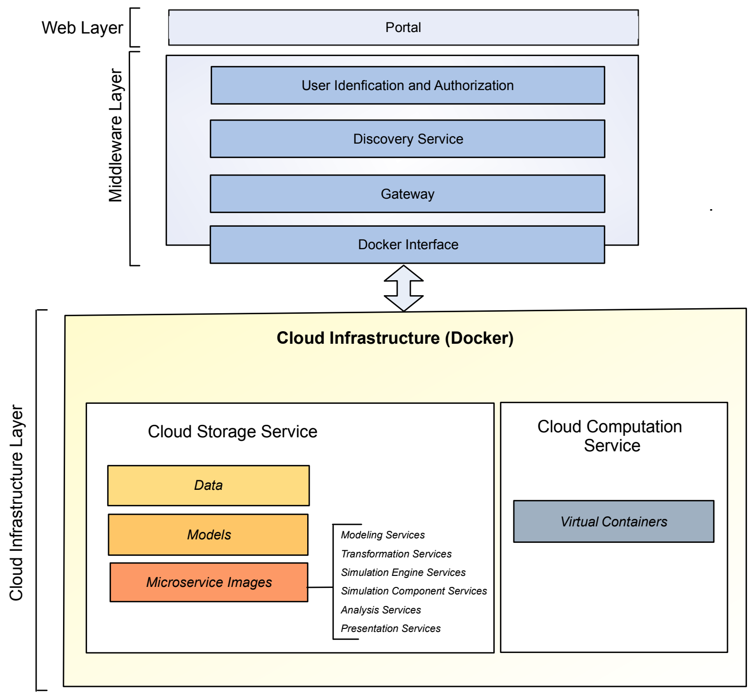

Figure 1: SOASim platform architecture.

- **Discovery Services**: component to retrieve services from the platform catalog.
- **Gateway**: component in charge of managing the communication and the cooperation among the several services that have to be orchestrated for executing a given simulation experiment.
- **Docker Infrastructure**: component that provides the services for interacting with the underlying cloud infrastructure by use of the Docker containerization approach. This component also provides the services for supporting the generation of scripts required for deploying and executing images on a Docker container.

According to the aforementioned perspective, the following categories of coarse-grained services are available:

- *Modeling services*, which provide visual environments to build simulation or system models.
- *Transformation services*, which automate the generation of models or implementations by use of model-to-model and model-to-text transformations, respectively. Transformation services can be used to generate executable simulation models, the code of a single simulation component, intermediate models at different levels of abstraction, simulation models using different formalisms, etc.
- *Simulation engine services*, which provide simulation engines to execute simulation models (e.g., a queueing network simulation engine).
- *Simulation component services*, which implement single fine-grained simulation components or primitives (e.g., a scheduler or a service center).
- *Analysis services*, which take as input the raw simulation results and carry out the output analysis.
- *Presentation services*, which are used to generate the visual representation of simulation results.

The next section illustrates how the SOASim platform can be used to implement, deploy and execute a simulation according to the fine-grained MSaaS paradigm.

4 MSAAS-ORIENTED SIMULATION DEVELOPMENT PROCESS

The simulation-based analysis of a given system is an effort that requires the cooperation of simulation-experts, domain experts and system engineers, in order to i) correctly identify the simulation objectives, ii) identify the required level of detail for the events of interest, and iii) correctly understand the behavior of the system, so to consistently specify the simulation requirements and the simulation model. In addition, the deployment and the execution of the simulation model implementation may require the availability of an execution platform that can be costly to be setup, especially in case of distributed simulation approaches. In this context, the adoption of the SOASim platform provides a valuable solution to:

- save the cost of setting up and maintaining a distributed network infrastructure to deploy the simulation model implementation;
- benefit from the SOASim Docker interface to properly manage the underlying cloud infrastructure;
- make use of the several SOASim (coarse- and fine-grained) services available in the catalog, which can be easily selected to maximize the software reuse;
- expose the simulation model implementation as an additional service offered by the SOASim platform.

The MSaaS process to build, deploy and execute a simulation experiment in SOASim consists of the following steps:

1. **Simulation Requirements Specification**: simulation engineers and system experts cooperate to define the system behavior that has to be simulated and the relevant level of abstraction.
2. **Simulation Model Specification**: several different languages and formalisms may be used to specify the simulation model, according to the intended use and the specific simulation requirements. As an example, this work makes use of queueing networks (QNs), a well-known formalism for system performance analysis (Sauer, MacNair, and Salza 1980).
3. **Simulation Components Identification**: the model specified at the step 2 results from the orchestration of simulation language primitives and components, which need to be properly identified.
4. **Microservice Discovery and Implementation**: according to the MSaaS paradigm, the primitives and components mentioned at the previous step are available as microservices. Different cases can be considered for each identified component:
   (a) the implementation of the component is not available. In this case the microservice must be implemented from scratch, and then included into the SOASim catalog for future use;
   (b) the implementation of the component is not available but a model transformation is available in the SOASim catalog. The microservice implementation can thus be obtained throughout the execution of the model transformation that takes as input a model of the component and yields as output its implementation;
   (c) an implementation is available as a standalone executable code. In this case the microservices is obtained by wrapping the business logic in a container;
   (d) the component behavior can be mapped to one of the existing SOASim services. In this case the service `URI` has to be retrieved and integrated in the simulation.
5. **Deployment**: each required service is deployed onto the SOASim cloud infrastructure.
6. **Simulation Execution**: finally, an orchestration service is deployed to properly manage the execution of the simulation model and complete the simulation-based analysis.

A simulation execution is obtained by orchestrating a set of containerized services deployed onto the cloud infrastructure and interacting by use of a common gateway. Some of such services are already existing in

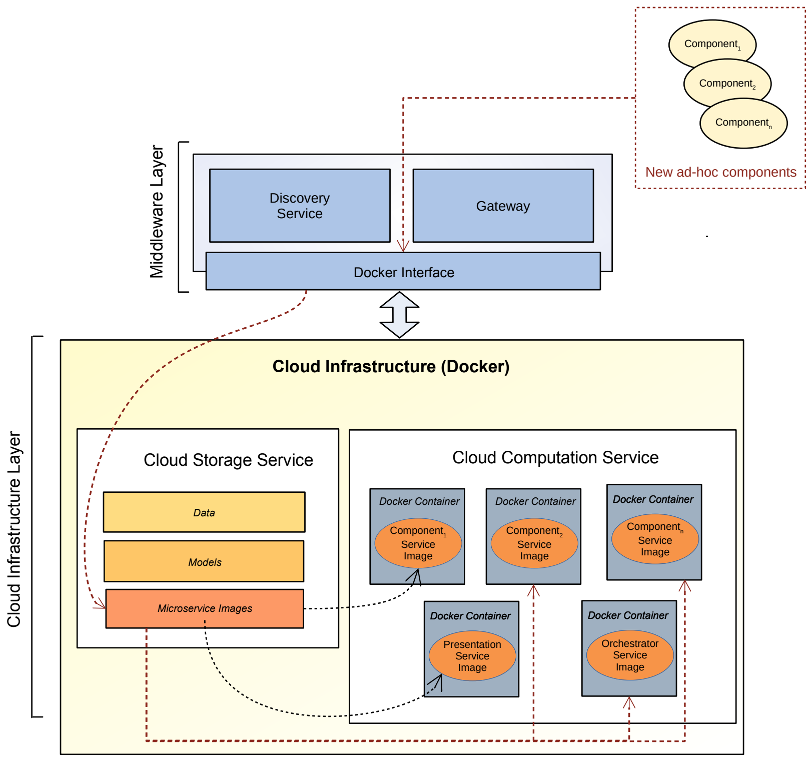

Figure 2: Integration of existing and ad-hoc services for simulation execution.

the SOASim catalog, so that related images can be retrieved from the cloud storage area and deployed in a container. Additional services, which are not available in the SOASim catalog, must be developed ad-hoc (either from scratch or by using transformation services), so that the required image can be created, stored onto the cloud-based storage area and eventually deployed in a container. Figure 2 illustrates an example use of SOASim to build and execute a simulation resulting from the integration of existing and ad-hoc developed services.

5 EXAMPLE APPLICATION

This section applies the process steps introduced in Section 4 to the development of a conventional DES based on the queueing network (QN) formalism, which targets performance-oriented metrics such as response time and throughput. Additional metrics can be taken into account by using alternative formalisms (Garro and Tundis 2012, Egert et al. 2018) The example application makes use of the *Spring framework* and its related technologies and libraries to implement the required components: *Eureka* for the discovery service, *Zuul* for the gateway, *Feign* to enable the interprocess communication, and *Ribbon* for load balancing (Pivotal Software 2019).

5.1 DES Requirements and QN Model specification

Figure 3 illustrates the considered QN model, which includes two service centers: `CPU` and `Input/Output`. Each job that leaves the source is initially routed to the `CPU` service center. Then, the job either leaves the networks or requires a service to the `Input/Output` center (to be eventually re-routed to the `CPU` service center).

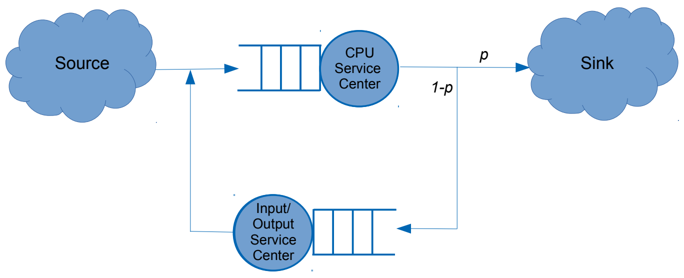

Figure 3: QN model for the example DES application.

The QN model in Figure 1 is implemented as a DES according to the event-based worldview, in which the scheduling and processing of asynchronous discrete events over time drives the simulation execution and the state evolution. In this respect, the following events are to be scheduled and processed for simulating the execution of the addressed QN model: CPU Job Arrival, CPU Service Completed, I/O Service Completed and Leave.

Additional details about the QN model (i.e., parameters such as number of users, inter-arrival times, service times, etc.) are not given here, being the paper focused on how to use the microservice-based features of SOASim to build, deploy and execute the model, rather than on the numerical solution of the simple QN model in Figure 3.

5.2 DES components Identification and Discovery

The building blocks of QN-based simulation models consist of the instances of language primitives (e.g., service centers, sources, etc.) and the DES components in charge of managing the simulation execution (e.g., clock, event scheduler, etc.).

In this respect, Figure 4 illustrates the DES logical architecture for the example application, along with the Spring-related technologies at the basis of each service implementation. The architecture consists of the following services:

- **Calendar Service**, which manages the list of events occurring during the simulation.
- **Scheduler Service**, which determines the next event that will occur and schedules the corresponding management.
- **Clock Service**, which stores the simulation time.
- **Gateway**, which enables the communications among services.
- **Analyzer Service**, which collects data during the simulation.
- **Orchestration Service**, which oversees the simulation execution and manages the simulation state.
- **Source**, which implements the simulation of the QN source node (CPU Job Arrival event management).
- **CPU Center**, which implements the simulation of the QN CPU service center node (CPU Service Completed event management).
- **Input/Output**, which implements the simulation of the QN I/O service center node (I/O Service Completed event management).
- **Sink**, which implements the simulation of the QN sink node (Leave event management).
- **Analyzer Service**, which takes as input a set of output data and generates the required reports to be displayed to users.

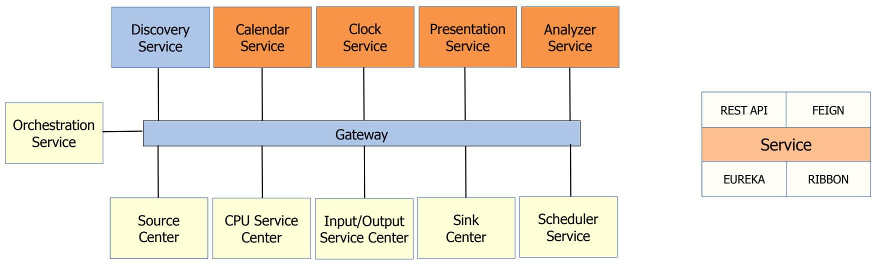

Figure 4: DES logical architecture, with details of a single component service.

Once the required service list is identified, the SOASim discovery service allows the user to retrieve the set of existing services to be reused and integrated, as well as the set of services that have to be implemented from scratch.

In the considered example application, it is assumed that the following services are already available in the catalog: *Calendar*, *Analyzer*, *Clock* and *Presentation*. Differently, the *Scheduler*, *Orchestration*, *Source Center*, *CPU Service Center*, *I/O Service Center* and *Sink Center* services have to be finely tuned with respect to the actual simulation objectives, and thus require an ad-hoc implementation. Figure 4 illustrates the DES logical architecture, with light blue boxes representing services provided by the SOASim platform, orange boxes services already available in the service catalog, and yellow boxes services to be implemented from scratch.

According to the Spring framework, each service exposes a REST API to be provided as a REST microservice, a connector to the discovery services (Eureka), the Feign interface for services communication, and the Ribbon library for load balancing.

5.3 Implementation, Deployment and Execution of DES services

The service discovery feature allows users to retrieve the set of already available services together with the information required for service invocation (i.e., the related *URI*). Differently, for the remaining services, which require an ad-hoc implementation, the corresponding REST microservices have to be developed and deployed onto the SOASim cloud infrastructure. In this respect, it is worth noting that the development of new microservices may take advantage of the existing transformation services, which support the automated generation of service executable code (e.g., in Java or C++).

As an example, let us consider the CPU service center. The SOASim infrastructure provides a Docker Interface component to manage the interaction with the underlying cloud-based infrastructure. In other words, SOASim allows to automate the generation of scripts and execution of corresponding commands to create a new image of the microservice and deploy it onto a computation node.

In this respect, Listing 1 shows the `Dockerfile` used to automate the steps to create an image (with the `ENTRYPOINT` instruction configuring the container to be run as an executable), while Table 1 summarizes the API providing the endpoints for accessing the operations provided by the service implementation.

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
COPY "target/cpu-service-1.0-SNAPSHOT.jar" "app.jar"
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-
jar","/app.jar"]
```

Listing 1: `Dockerfile` for creating the image of the CPU service center.

Table 1: REST API for the CPU service center.

Method	Endpoint	Property	Description
GET	/	-	Returns the service state
GET	/queue/size	-	Returns the CPU queue length
POST	/queue/add	Job object in JSON format	Add a job to the CPU queue
POST	/service/add	Job object in JSON format	Add a job to the CPU service center
GET	/service/time	-	Returns the service time
GET	/service/status	-	Returns the CPU service state (busy or idle)
POST	/routine	-	Executes the CPU end service routine
POST	/service/stable	-	Saves the current service state as stable state
PUT	/service/stable	-	Use the last stable state as current state
POST	/service/reset-status	-	Reset the CPU center to the initial state

The final step of the process described in Section 4 consists in the execution of the various DES services deployed on the MSaaS platform. In this respect, the SOASim platform allows both to automate the simulation orchestration (e.g., the execution and the management of the various services) and to expose the overall simulation as an additional service to be reused in different operational contexts.

According to the stack of technologies at the basis of SOASim, the orchestration of the microservices implementing the simulation components is carried out by use of Docker Compose, which provides a tool for configuring, starting and managing software applications consisting of a set of Docker images.

The services configuration is serialized in a YAML file, which allows using a single command to execute the whole set of configured services. A fragment of the YAML for the example application is provided in Listing 2, in which the `build` instruction for the `cpu-service` defines the path to reach the `Dockerfile` illustrated in Listing 1, and thus create an image with name specified by the `image` instruction.

```
version: '2'
services:
registry-service:
...
cpu-service:
  container_name: cpu-service
  build: ./cpu-service
  image: msaas/cpu-service:latest
  links:
    - registry-service:registry-service
  depends_on:
    - registry-service
  networks:
    - msaas-QN
...
networks:
  msaas-QN:
   driver: bridge
```

Listing 2: Docker Compose configuration file in YAML (fragment).

6 CONCLUSIONS

The paper has illustrated the architecture of an MSaaS platform that integrates fine-grained M&S services. The proposed platform, named SOASim, hides the complexity of the underlying cloud-based infrastructure and supports the discovery and the integration of existing services, as well as the containerization and the deployment of new ones.

The introduction of fine-grained M&S services in MSaaS platforms, which typically provide coarse-grained M&S services, obtains several advantages in terms of modularity, reusability and maintainability.

The finer granularity offered by the proposed approach also opens the way to an enhanced interoperability among different formalisms or domain-specific simulation languages, so to support the definition and implementation of hybrid simulation solutions. In addition, the availability of already existing coarse-grained services allows to further enhance the benefits of the proposed approach, as outlined in the paper.

The paper has introduced a simple example application of SOASim to the development of a queueing-based discrete event simulation. The prototype Docker-based implementation of SOASim has been used to assess the feasibility and the effectiveness of the proposed approach.

Ongoing work includes i) the evaluation of the scalability of microservice-based simulation approaches by use of advanced orchestration systems for automating application deployment, scaling, and API gateway management, such as Kubernetes or Mule API Gateway, and ii) the SOASim extension to provide access to already existing domain-specific simulation components as microservices, so to show the potential of the proposed approach in significant application domains, such as in cyber physical systems engineering.

REFERENCES

Biagini, M., F. Corona, M. Picollo, M. L. Grotta, J. Jones, A. Scaccianoce, A. Mursia, C. Faillace, and D. Prochazka. 2017. "Modeling and Simulation as a Service from End User Perspective". In *Proceedings of the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC)*.

Bocciarelli, P., A. D'Ambrogio, A. Mastromattei, E. Paglia, and A. Giglio. 2017. "Business process modeling and simulation: State of the art and MSaaS opportunities". In *Proceedings of the 2017 Summer Simulation Multi-Conference (SummerSim 2017)*, edited by S. E. and D. A. The Society for Modeling and Simulation International, The Society for Modeling and Simulation International.

Cayirci, E. 2013. "Modeling and simulation as a cloud service: a survey". In *Proceedings of the 2013 Winter Simulation Conference*, WSC '13, pp. 389–400. San Diego, CA, USA, Society for Computer Simulation International.

D'Ambrogio, A., P. Bocciarelli, and A. Mastromattei. 2016, 12. "A PaaS-based framework for automated performance analysis of service-oriented systems". In *Proceedings of the 2016 Winter Simulation Conference*, pp. 931–942.

Daya, S., N. Van Duy, K. Eati, C. M. Ferreira, D. Glozic, V. Gucer, M. Gupta, S. Joshi, V. Lampkin, M. Martins et al. 2016. *Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach*. IBM Redbooks.

Docker 2016. "Docker - Build, Ship, and Run Any App, Anywhere". Website: https://www.docker.com. Accessed May 27, 2019.

Egert, R., A. Tundis, S. Roth, and M. Mühlhäuser. 2018. "A Service Quality Indicator for Apriori Assessment and Comparison of Cellular Energy Grids". In *International Conference on Sustainability in Energy and Buildings. KES-SEB 2018*, Volume 13, pp. 322–332. Springer.

Garro, A., and A. Tundis. 2012. "Modeling and simulation for system reliability analysis: The RAMSAS method". In *2012 7th International Conference on System of Systems Engineering (SoSE)*, pp. 155–160. IEEE.

Gianni, D., A. D'Ambrogio, and A. Tolk. (Eds.) 2014. *Modeling and Simulation-Based Systems Engineering Handbook*. CRC Press.

Hayes, B. 2008, July. "Cloud computing". *Commun. ACM* vol. 51 (7), pp. 9–11.

Karmel, A., R. Chandramouli, and M. Iorga. 2016. *NIST Special Publication 800-180: NIST Definition of Microservices, Application Containers and Virtual Machines*.

Mell, P. M., and T. Grance. 2011. "The NIST Definition of Cloud Computing". Technical report, Gaithersburg, MD, United States.

Menascé, D. A., and P. Ngo. 2009. "Understanding Cloud Computing: Experimentation and Capacity Planning". In *Proceedings of the 2009 Computer Measurement Group Conference*.

Newman, S. 2015. *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.".

Papazoglou, M. P., and D. Georgakopoulos. 2003, October. "Service-oriented computing". *Commun. ACM* vol. 46, pp. 24–28.

Pivotal Software 2019. "Spring Framework". Website: https://spring.io/. Accessed May 27, 2019.

Sauer, C. H., E. A. MacNair, and S. Salza. 1980. "A language for extended queueing network models". *IBM Journal of Research and Development* vol. 24 (6), pp. 747–755.

Siegfried, R., J. Lloyd, and T. V. D. Berg. 2018. "A New Reality: Modelling & Simulation as a Service". *Journal of Cyber Security and Information Systems* vol. 6 (3), pp. 18 – 29.

Sokolowski, J., and C. Banks. 2009. *Principles of Modeling and Simulation: A Multidisciplinary Approach*. Wiley.

Taylor, S. J., T. Kiss, A. Anagnostou, G. Terstyanszky, P. Kacsuk, J. Costes, and N. Fantini. 2018. "The CloudSME simulation platform and its applications: A generic multi-cloud platform for developing and executing commercial cloud-based simulations". *Future Generation Computer Systems* vol. 88, pp. 524 – 539.

Vaquero, L. M., L. Rodero-Merino, J. Caceres, and M. Lindner. 2008, December. "A break in the clouds: towards a cloud definition". *SIGCOMM Comput. Commun. Rev.* vol. 39 (1), pp. 50–55.

Yamazaki, T., H. Ikeno, Y. Okumura, S. Satoh, Y. Kamiyama, Y. Hirata, K. Inagaki, A. Ishihara, T. Kannon, and S. Usui. 2011. "Simulation Platform: A cloud-based online simulation environment". *Neural Networks* vol. 24 (7), pp. 693 – 698.

AUTHOR BIOGRAPHIES

PAOLO BOCCIARELLI is a postdoc researcher at the University of Rome Tor Vergata (Italy). His research addresses the application of M&S and model-driven development to software and systems engineering and business process management. His email address is paolo.bocciarelli@uniroma2.it.

ANDREA D'AMBROGIO is associate professor at the University of Roma Tor Vergata (Italy). His research interests are in the fields of model-driven software engineering, dependability engineering, distributed and web-based simulation. His email address is dambro@uniroma2.it.

ANDREA GIGLIO is assistant professor at the "Guglielmo Marconi" University. His research interests include model-driven system and software engineering and business process management. His email address is a.giglio@unimarconi.it.

EMILIANO PAGLIA is a postdoc researcher at the University of Rome Tor Vergata (Italy). His research interests include model-driven engineering and business process modeling and analysis. His email address is emiliano.paglia@uniroma2.it.