Proceedings of the 2023 Winter Simulation Conference
C. G. Corlu, S. R. Hunter, H. Lam, B. S. Onggo, J. Shortle, and B. Biller, eds.

PROJECT SIMULATION, VALIDATION AND DEPLOYMENT WITH DEVS. IOT FRAMEWORK FOR BLOOMS MONITORING AND ALERT

Segundo Esteban
José L. Risco-Martín
Jesus Chacon
Eva Besada-Portas

ACYA Department
Universidad Complutense de Madrid
Fac. CC. Fisicas, Plaza de Ciencias
Madrid 28040, SPAIN

Giordy A. Andrade

ISCAR Group
Universidad Complutense de Madrid
Fac. CC. Fisicas, Plaza de Ciencias
Madrid 28040, SPAIN

ABSTRACT

Harmful Algal and Cyanobacterial Blooms (HABs) constitute a relevant public health and ecological hazard due to their frequent production of toxic metabolites, which is increased by the current vulnerability of water resources to environmental changes such as global warming, population growth, and eutrophication. These blooms have been typically assessed by combining predictive models with manual collection. However, these processes are generally independent and do not provide data with sufficient resolution to apply proactive policies. In this work, we propose a novel and integrative framework to straightforwardly combine the conception, design, and deployment of advanced Early-Warning Systems (EWSs) that will allow us to automate all the processes involved in HABs detection and management and apply proactive policies. The framework is built upon solid Modeling and Simulation (M&S) principles, through Model Based Systems Engineering (MBSE) as the driving methodology and Discrete Event System Specification (DEVS) as the M&S formalism.

1 INTRODUCTION AND RELATED WORK

Water scarcity affects about 4 billion people, two-thirds of the world's population (Mekonnen and Hoekstra 2016). Additionally, meeting water quality standards is becoming increasingly difficult due to population growth, climate change, pollution, and overexploitation, which introduce undesirable substances or microorganisms in the water bodies that can become a health risk.

One important source of those substances is in Harmful Algal and Cyanobacterial Blooms (HABs) due to their frequent production of toxic secondary metabolites. HABs typically appear in various freshwater ecosystems like reservoirs, lakes, and rivers (Vincent 2009). Their intensity and frequency have increased globally during the last decade, mainly due to the current vulnerability of water resources to environmental changes, such as global warming, population growth, and eutrophication. It is worthwhile to mention that the danger is not confined to the immediate area surrounding the water source since extracellular material from freshwater HABs has been observed in the water and the atmosphere at locations far beyond their edges (Schmale et al. 2019).

Efforts are being made to detect these organisms, and the most common method involves taking samples manually by specialized personnel and analyzing them in the laboratory. In some uncommon cases, automatic instruments placed at fixed locations may acquire data and samples. Financial and personnel resource restrictions reduce the manual collection to the moments of the year when HABs are more likely to appear at a few geographical points and with minimal frequencies. The delay suffered by analytical

results and the limited capacity to interpret the current scenario reduces the prediction, prevention, and mitigation capability of the authorities responsible for the distribution of drinking water and its recreational uses (Meriluoto et al. 2017). This is critical when deploying Early-Warning Systems (EWSs), whose essential work is to collect water samples and identify the cyanobacterial cell or algae density as soon as possible. Creating new monitoring and early detection systems is necessary to forecast the formation and toxin production of HABs without the abovementioned barriers. These systems can assist water managers and authorities implement proactive policies safeguarding public health.

In this context, Modeling and Simulation (M&S) can be used to clarify the dynamics of HABs. For instance, numerical-based and data-driven machine learning models have been extensively used to simulate HABs in aquatic systems (Long et al. 2011),(Pyo et al. 2019). These techniques try to reach accurate predictions through mathematical models. These models have been integrated into more generic software tools like the EE Modeling System (EEMS) (DSI 2022). Based on these models and tools, various countries have attempted to build EWSs with the support of predictive systems (Wu et al. 2022).

Our approach is oriented to a system of systems architecture, a more holistic and integrative model that includes the use of the mathematical models mentioned above, and a modern model of the automated infrastructure of the EWS.

Our research project develops an automated HAB monitoring and alerting system, which merges automated data collection and simulation of parameterized models. Due to the system's complexity, it has been designed in layers according to the Internet of Things (IoT) paradigm. Figure 1 shows a high-level representation of the main components of the system. The Unmanned Surface Vehicles (USVs), sensors, and weather stations reside in the Edge-Things layers. These elements are in charge of measuring the state of the water body and are managed and controlled through the Ground Control Station (GCS), which is housed in an intermediate layer called Fog. The GCS uses inference services to predict the HABs and control the USVs so that the sensors measure in the right places. Finally, all the information is stored on servers residing in the Cloud layer, where reports and alerts should be generated for water managers. Cloud servers also train inference and alert models using historical data. Note that the exchange of information, control, and training loops between the different layers must be closed.

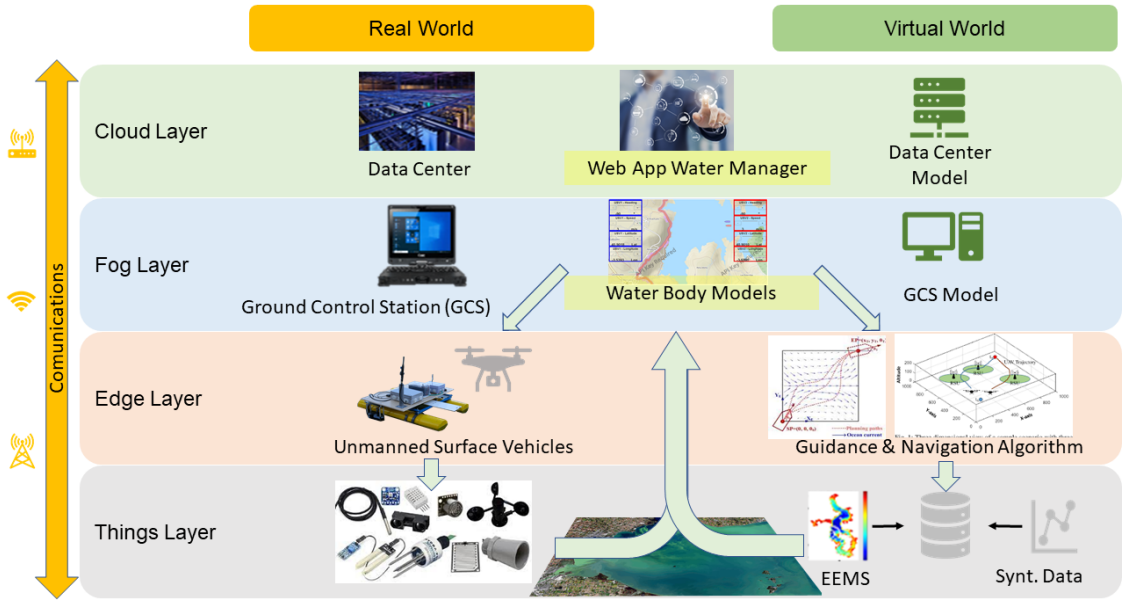

Figure 1: Architecture of the EWS monitoring and alerting project divided into IoT layers: Cloud, Fog and Edge-Things.

The development of an advanced EWS is a complex and multidisciplinary project, which is distributed among teams in charge of different subsystems or units:

- Modelling: Modelling and inference of biodynamics of water masses and HABs, where part of the team are biologists.
- Instrumentation: Development of sensors and communications.
- Ships: Guidance, Navigation and Control (GNC) of USVs (Ferrero-Losada et al. 2023).
- *IoT* Software: Simulator, software, and system integration (Herguedas-Pinedo et al. 2023).

As the project has a research nature, the subsystems are unavailable during its development, as they are being designed in parallel. For this reason, the IoT software team has developed a simulator of the project, which covers all aspects: user, hardware, software, communications, data, hydrodynamic and biological environment, etc. The first objective is to determine the models that make up the system and define its interface so that it will later be possible to integrate all the subsystems. In addition, this simulator will make it possible to validate and deploy the different subsystems in a distributed manner, making it possible to deal with the time lag between the working groups. To deal with this issue, three development stages are distinguished for all the subsystems, as shown in figure 2:

- Simulation: In this stage, the interfaces of the subsystems are defined and integrated into a project simulator using synthetic data stored in files. The objective is to define the logic and format of the messages. At this stage, the functionality of the subsystems is not yet implemented, so it is impossible to close the control and training loops.
- Validation: At this stage, the subsystem to be validated must be fully functional, although other subsystems can be roughly simulated. Realistic data, emulated through EEMS, are used, allowing control loops to be closed and inference models to be used and trained.
- Deployment: At this stage, the subsystem to be deployed must run on real hardware and use real measurements. During partial deployment, the deployed subsystem can be evaluated against other subsystems in the validation stage. When all subsystems are deployed, the software integrating them will be an evolution of the simulator running in real-time.

In closed-loop systems, control loops are required to generate stable outputs before deployment, to avoid system damage due to instabilities. In addition, in this case, inference models are part of the control loop. Control engineers are concerned about the stability of intelligent controls. Much of their efforts are focused on ensuring their stability, as can be seen in (Blondin et al. 2019) and (Balakrishnan et al. 2008). It is strictly necessary that these models are trained before the system is deployed. Therefore, they are trained offline during the validation stage using simulations. Later, these models will be re-trained online with actual data at the system deployment and operation stage.

We tackle this complex system using well-grounded M&S methodologies. As a result, we apply the principles of Model Based Systems Engineering (MBSE): (i) model-based since MBSE is based on the use of models to represent and manage information about a system, (ii) system-centric, focusing on the system as a whole, (iii) iterative and incremental process, which involves the development of models over time, (iv) collaboration between stakeholders, including system engineers, domain experts, etc., (v) traceability between requirements, design, and implementation, (vi) reuse of models, components, and other artifacts to improve efficiency and reduce the risk of errors, and (vii) verification and validation to ensure that the system meets its requirements and that it operates as intended (Wymore 2018). In our case, MBSE is driven by Discrete Event System Specification (DEVS) (Zeigler et al. 2018). DEVS is a modular and hierarchical formalism for modeling discrete event systems based on set theory. It includes two types of models, atomic and coupled. Atomic models define the system's behavior, whereas coupled models are the aggregation/composition of two or more models (atomic or coupled) connected by explicit couplings. This makes DEVS closed under coupling and allows us to use networks of systems as components in larger

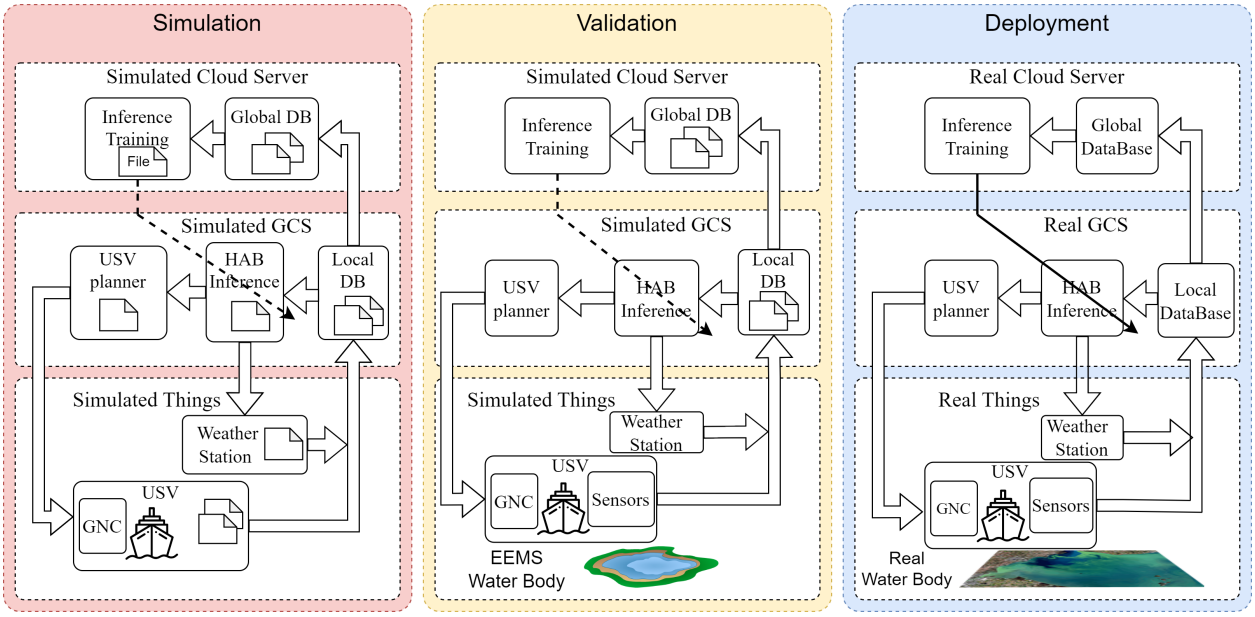

Figure 2: Stages of development and control loops.

coupled models, leading to hierarchical and modular designs. Overall, DEVS provides a framework for information modeling with several advantages in the analysis and design of complex systems: completeness, verifiability, extensibility, and maintainability.

Our EWS is formed by several subsystems that can be in different development stages. Some subsystems will be in the simulation stage, others in the validation stage, and others in the deployment stage. To do so, we start with a DEVS simulator implemented with the xDEVS simulation engine (Risco-Martín et al. 2022), which captures the definition of the project by IoT layers. This simulator will define the functional and extra-functional model of the different subsystems, allowing us to replace simulated virtual subsystems with actual subsystems to verify their behavior integrally. Each work team can validate a given subsystem against the rest of the subsystems, which will work in a simulated way, allowing a distributed validation. The objective is that the initial simulator evolves towards the final deployment of the system through a progressive substitution of simulated (virtual) elements by actual elements.

The simulator has been developed in Python due to the facilities provided by this language for data handling. The xDEVS Application Programming Interface (API) is used, which is multiplatform, multilanguage, and parallelizable. The xDEVS implementation allows working in simulated time to perform functional verifications and in soft real-time to perform extra-functional validations.

2 DEVELOPMENT PROCESS

2.1 Simulation Stage

The development process starts with designing a simulator architecture that integrates all subsystems, including algorithms, software, and hardware. Following the MBSE methodology, the subsystems must first be modeled at a very high level to define their interfaces and information flow. For this purpose, the system has been divided into the three typical IoT layers: Cloud, Fog, and Edge. Figure 3 shows the DEVS architecture of the simulator for a bloom monitoring case, where a USV and a weather station are used as data sources.

The Cloud layer hosts web services, historical databases, water body simulations, prediction models, and high-level computation. The training or tuning of the inference models is also performed in this layer. Weather forecasts, used to predict future blooms, also come from cloud servers.

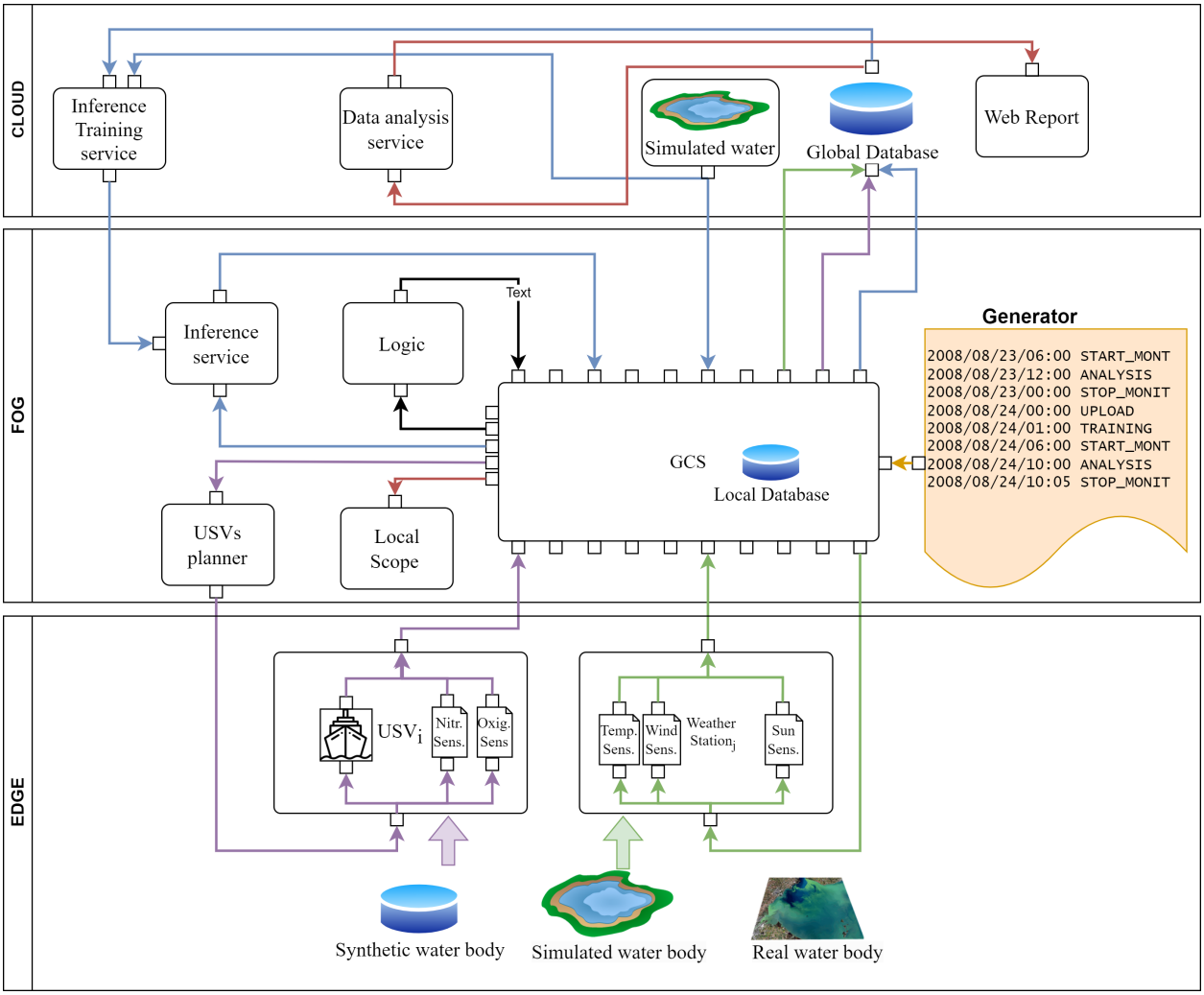

Figure 3: Architecture of the blooms monitoring and warning project simulator.

The Fog layer is responsible for the computation at the water body level, where an operator commands the GCS through the *Generator* module. In this layer, services are used to solve part of the work: a logic service defines the mode of operation, an inference service predicts the evolution of blooms, and a planning service commands the USVs. In addition, the operator analyses the results through the *Local Scope* module.

The Edge layer contains Things: sensors, ships, actuators, etc. Some with edge computing capabilities, such as, for example, the USVs, which must navigate to waypoints and take measurements using onboard sensors. This layer also includes other sensors, e.g., weather stations.

The top level of the simulator implements the layer hierarchy on a coupled model. The Cloud layer is linked to several Fog layers, one for monitoring each water body. The Fog layer can contain multiple zones of water to be monitored, multiple *USVs*, and multiple sensors, as shown in the following code.

```
class BloomsMonitorModel(Coupled):
    def __init__(self, name, commands_path: str):
        # COMMANDER: Control de simulation and emulate the user
        commander = Commander("Generator", commands_path)
        # CLOUD SERVER: Storage of water bodies data and training.
        cloud = Cloud("Cloud", num_water_bodies=2)
        # FOG SERVER 1: Manager of Water Body 1
```

```
fog1 = FogServer("FogServer1", n_uav=1)
# UAV 1, CianoSensor measures the Bloom and SunSensor the Sun radiation.
ship1 =  FileIn("ShipPos1", "data/USV1.csv",DataEventId.POS3D)
ciansensor1 = FileIn("CianSen1", "data/Bloom1.csv",DataEventId.BLOOM)
oxisensor1 = FileIn("OxiSensor1","data/Bloom1.csv",DataEventId.OXIGEN)
sunsensor1 = FileIn("SunSen1", "data/Weather1.csv",DataEventId.SUN)
# FOG SERVER 2:  Manager of Water Body 2 with 2 USVs
fog2 = FogServer("FogServer2", n_uav=2)
ship21 =  FileIn("ShipPos21", "data/USV21.csv",DataEventId.POS3D)
ship22 = FileIn("ShipPos22","data/USV22.csv",DataEventId.POS3D)
...
self.add_component(...)
...
self.add_coupling(..., ...)
...
```

The input of information to the system is mainly via the Edge layer. Measurements or data must be provided by the USVs, weather stations, and other sensors, either on their initiative or external request. This data must be propagated and processed through the layers. A generic event type encapsulated in the class *Event* is used for this purpose. All ports of the models will use this event type.

```
class Event:
    #A message to model events.
    id: str
    source: str
    timestamp: datetime = field(default_factory=datetime.now)
    payload: dict = field(default_factory=dict)
```

The fields of the class *Event* are:

- *id* is the identifier of the event.
- *source* is the generator of the event. *timestamp* is the date and time when the event was generated.
- *payload* is the event information. The event information is a Python dictionary, which allows a lot of versatility to contain multiple measurements of different types.

Different models have been used to inject events into the system during the project's development, from a simple file containing the sequence of events to servers responding to remote requests. As all the models use the same message format, they can all be connected indifferently to the simulator.

As shown in the codes, synthetic files (*FileIn*) have been used to perform the first tests of event propagation through the model hierarchy. These files can inject sensor measurement events that request measurements from the sensors or events that propagate processed signals, as seen in the *Simulation* stage of figure 5.

The sensors that can be used in this project have very different behaviors, so the following state logic has been developed for them:

- *OFF*: Standby, waiting for a request.
- *INIT*: Initialisation, time synchronization, and sending sensor info.
- *ON*: Initialised, waiting for a request.
- *WORK*: Taking a measurement.
- *DONE*: Sending the measurement.

With these simple elements, it has been possible to verify that the information is correctly propagated through the different layers and models of the simulator, generating the messages at the correct time.

2.2 Validation Stage

The EEMS tool has been used to work with realistic signals. This tool simulates the hydrodynamic and biological behavior of a body of water. In particular, Lake Washington has been modeled because it is very well parameterized. The *SimBody* service has been developed, which simulates a body of water on the output file of EEMS. This service provides measurements of all kinds of signals at any time and place, allowing the construction of simulated sensors, *SimSensor*. It is used in different layers of the simulator: in the Cloud layer to train the inference models, in the Fog layer to represent the information, and in the Edge layer to implement simulated sensor models. The simplicity of the code needed to build a simulated sensor based on this service is shown below.

```
bodyfile: str = "./data/Washington-1m-2008-09_UGRID.nc"
simbody: SimBody = SimBody("SimWater", bodyfile)
sensor_info_n = SensorInfo(id=SensorEventId.NOX,
                 description="Nitrogen␣Sensor␣(mg/L)",
                 delay=6, max=0.5, min=0.0, precision=0.1,
                 noisebias=0.01, noisesigma=0.001)
sensor_n = SimSensor("SimSenN", simbody, sensor_info_n)
```

The output files of EEMS are large, of several GiBs. A remote version of the *SimBody* service called *RestBody* has been generated to not saturate personal computers. This new service is hosted on a remote server with several TiB of hard disk, which can store simulations of multiple water bodies. The service is built on top of *FLASK* (Grinberg 2018), and queries are performed in *JSON* format (Crockford and Morningstar 2017). The programming is entirely transparent for the user, as in practice, this service is initialized similarly to *SimBody*, as shown in the following code.

```
simbody: SimBody =  RestBody("SimWater",
                    host="http://pc-iscar.dacya.ucm.es:5000",
                    bodyfile= "Washington-1m-2008-09_UGRID.nc")
```

Several subsystems have been validated using simulated water masses, simulated sensors, and simulated USVs. As the EEMS simulation of the entire water body is available, an augmented representation of the simulation can be made, showing the values recorded by the sensors and inference models on EEMS-generated signal maps. Figure 4 shows the state of the system during a bloom monitoring using a USV.

Using this framework, the subsystems that close the control and training loops have been validated:

- On the Cloud layer, a training service for bloom inference models.
- On the Fog layer, a logic for monitoring a water body, a bloom inference service, and the *Generator* module.
- On the Edge layer the GNC of a simulated USV and a sensor logic.

2.3 Deployment Stage

Finally, when a subsystem has been fully developed, it must be deployed on an evolution of the simulator running in real-time. Figure 5 shows a diagram of the project's current state, where some subsystems are in the simulation stage, others in the validation stage, and some in the deployment stage.

A commercial GPS sensor is used and a version of its drivers is already deployed and running in real-time. The simulator has been extended with a model, *Injector*, which allows external events to be injected in soft real-time. This model launches a parallel thread containing a Flask server. The GPS sends the available measurement to the Flask server in JSON format, and the *Injector* model inserts an event with the measurement into the simulator. A trace of a GPS software driver is shown below. When a message arrives from the GPS, the *Injector* generates an event with the information sent by the GPS. The event is then processed if it contains a message in NMEA format. It can be seen that the GPS is sending other different messages, in which case they are discarded. One problem that this real-time test points out is that

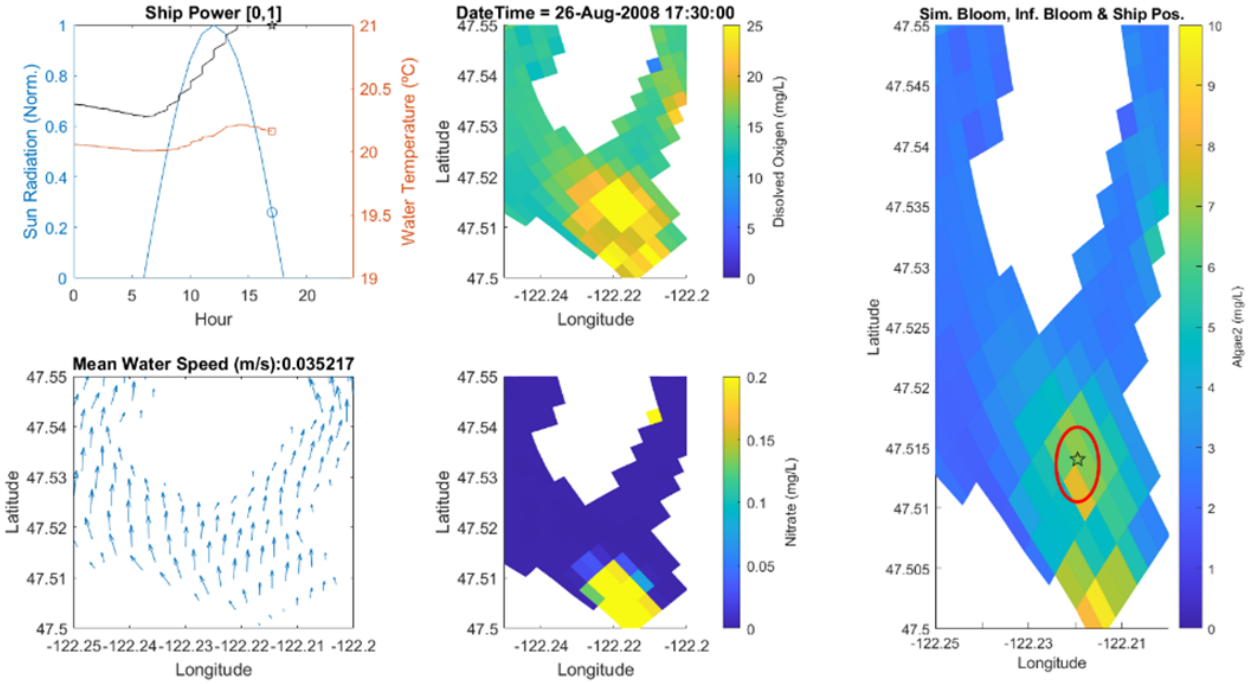

Figure 4: Augmented representation during a bloom monitoring simulation. The upper-left plot shows information on solar radiation, USV energy state and water temperature. The bottom-left plot shows an EEMS map of water surface velocity. The middle two plots show EEMS maps of oxygen and nitrate density in the surface layers. The right plot shows an EEMS map of the algal density in the surface layer, the position of the USV (black star) and the position and size of the inferred bloom (red circle).

the GPS time is 2 hours and 1 second out of phase with the computer time. The 2h lag is due to the fact that the computer is using the local time. The 1 second lag is due to the fact that UTC time is corrected by 1 second every few years, due to the slowing of the earth's rotation.

```
...
Starting event 8 @ t = 12:12:17
Starting event 9 @ t = 12:12:17
event 8 finished @ t = 12:12:17
INFO:__main__:event 8
{
 'timestamp': datetime.time(10, 12, 18),
 'latitude': 40.45073216666667, 'longitude': -3.726055166666667,
 'horizontal_dil': '1.42',  'num_sats': '07', 'gps_qual': 1
}
...
Starting event 14 @ t = 12:12:18
Starting event 15 @ t = 12:12:18
event 14 finished @ t = 12:12:18
INFO:__main__:event 14
{
 'timestamp': datetime.time(10, 12, 19),
 'latitude': 40.450736166666665, 'longitude': -3.7260635,
 'horizontal_dil': '1.42', 'num_sats': '07', 'gps_qual': 1
}
```

```
...
Starting event 19 @ t = 12:12:19
Starting event 20 @ t = 12:12:19
event 19 finished @ t = 12:12:19
INFO:__main__:event 19
{
 'timestamp': datetime.time(10, 12, 20),
 'latitude': 40.4507385, 'longitude': -3.726068166666667,
 'horizontal_dil': '1.42', 'num_sats': '07', 'gps_qual': 1
}
...
```

In the Cloud layer, a web service has been deployed that provides alerts and reports. Services launched after each day's simulation generate these alerts and reports. Figure 6 shows the view provided to the

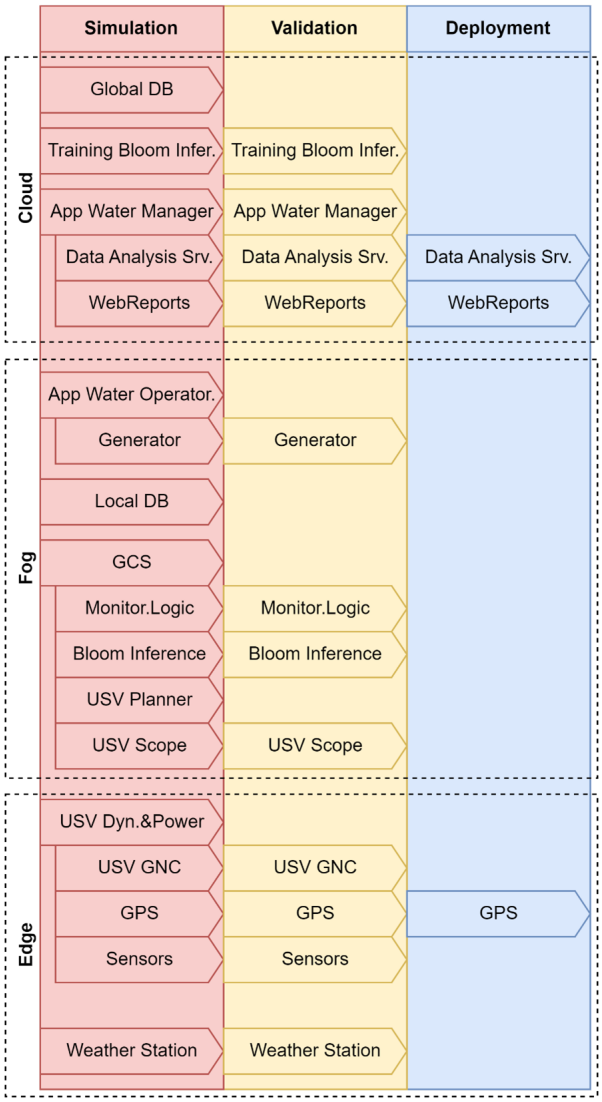

Figure 5: Status of development of the different subsystems/units of the project.

Water Manager. It shows the last monitored Bloom alerts and a map that allows locating the measurements performed by the USVs in the affected areas.

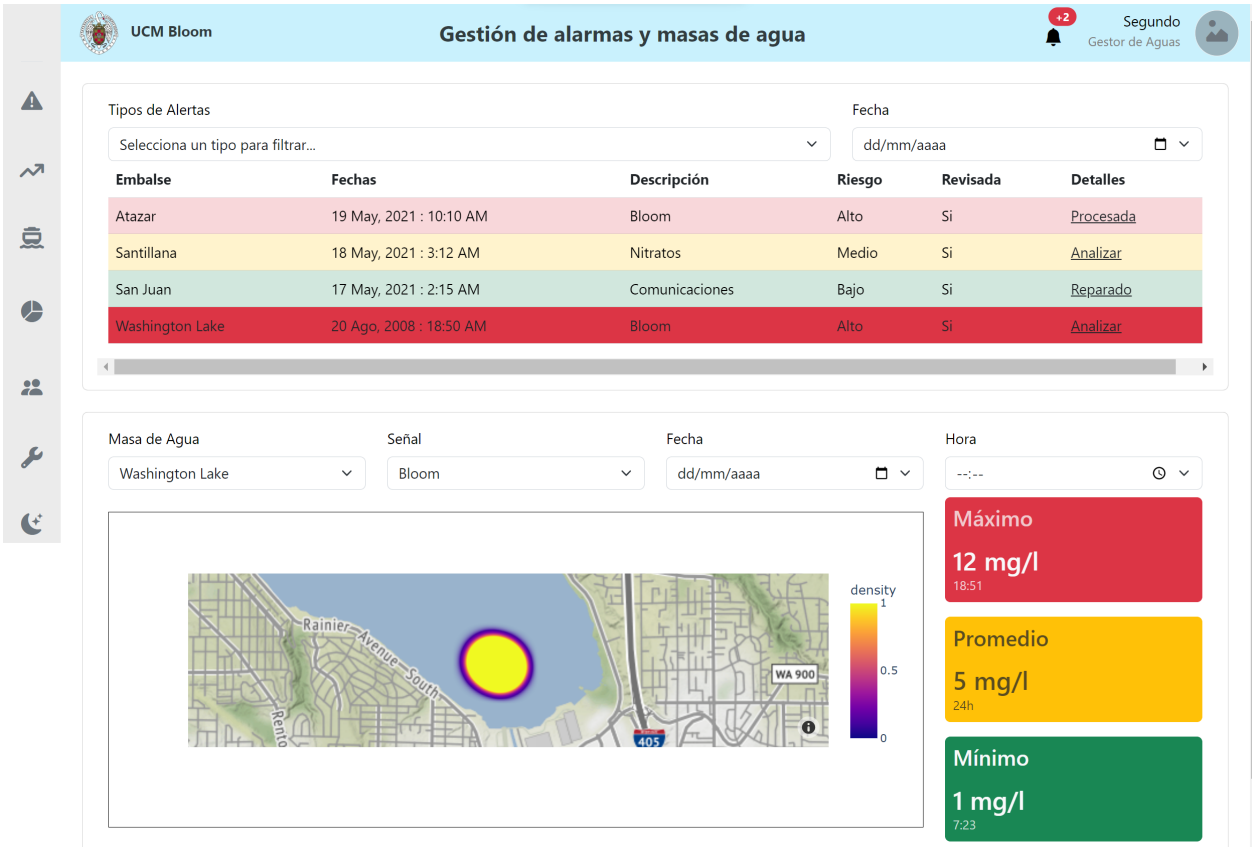

Figure 6: View of the alerts web page for the water manager.

Other views allow us to see the recorded and inferred signals and the status of the USVs sensors in detail.

3 CONCLUSIONS

HABs have become an important risk for recreational use and human consumption of water ecosystems like reservoirs, lakes, and rivers. Some efforts are being made to detect these organisms. However, these approaches are based on implementing independent reactive policies like manual sampling, automatic sampling in fixed locations, or using classic predictive models. New monitoring and early detection systems must be implemented to design a proactive, modern, automated, and integrative EWS.

In this paper, we have proposed the design of an advanced EWS. To tackle such a complex system, we have presented the implementation of solid MBSE principles driven by DEVS, which allows us to conceive, develop, validate, and straightforwardly deploy these complex control systems. As the different subsystems' development evolves, it is necessary to work with simulations of them, allowing us to validate and deploy the different subsystems in a distributed way. Moreover, the system can only be deployed once the control loops are stable. Therefore, it is crucial to have a system simulator to train the inference models that close the control loop.

Due to the complexity of this system, it has been divided into layers according to the IoT paradigm: Cloud, Fog, and Edge. A structured framework has been developed in layers that allow the development of the subsystems in stages: simulation, validation, and deployment.

As an application case, an EWS has been developed for inland waters. For this purpose, a simulator has been used to define the models' hierarchy, connection, and interfaces. Subsequently, the simulator has been provided with realism, using simulated sensors. This has allowed us to validate the functionality of some subsystems. Finally, the simulator has evolved towards real-time to allow the deployment of the subsystems.

ACKNOWLEDGMENTS

This work has been supported by the Research Projects IA-GES-BLOOM-CM (Y2020/TCS-6420) of the Synergic program of the Comunidad Autónoma de Madrid, SMART-BLOOMS (TED2021-130123B-I00) funded by MCIN/AEI/10.13039/501100011033 and the European Union NextGenerationEU/PRTR, and INSERTION (PID2021-127648OB-C33) of the Knowledge Generation Projects program of the Spanish Ministry of Science and Innovation.

REFERENCES

Balakrishnan, S. N., J. Ding, and F. L. Lewis. 2008. "Issues on Stability of ADP Feedback Controllers for Dynamical Systems". *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 38(4):913–917.

Blondin, M. J., J. Sanchis Sáez, and P. M. Pardalos. 2019. *Control Engineering from Classical to Intelligent Control Theory—An Overview*, 1–30. Cham: Springer International Publishing.

Crockford, D., and C. Morningstar. 2017. "Standard ECMA-404 The JSON Data Interchange Syntax". *Geneva: ECMA International*.

DSI 2022, January. "EE Modeling System". https://www.eemodelingsystem.com.

Ferrero-Losada, S., E. Besada-Portas, J. L. Risco-Martín, and J. A. López-Orozco. 2023. "DEVS-Based Modeling and Simulation of Data-Driven Exploration Algorithms of Lentic Water Bodies with an ASV". In *Proceedings of the 2023 Annual Modeling and Simulation Conference (ANNSIM'23)*.

Grinberg, M. 2018. *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.

Herguedas-Pinedo, B., J. L. Risco-Martín, S. Esteban, J. A. López-Orozco, and E. Besada-Portas. 2023. "Predictive Modeling and Simulation System for the Management of Harmful Cyanobacteria Blooms". In *Proceedings of the 2023 Annual Modeling and Simulation Conference (ANNSIM'23)*.

Long, T.-Y., L. Wu, G.-H. Meng, and W.-H. Guo. 2011. "Numerical Simulation for Impacts of Hydrodynamic Conditions on Algae Growth in Chongqing Section of Jialing River, China". *Ecological Modelling* 222(1):112–119.

Mekonnen, M. M., and A. Y. Hoekstra. 2016. "Four Billion People Facing Severe Water Scarcity". *Science Advances* 2(2):e1500323.

Meriluoto, J., L. Spoof, and G. A. Codd. 2017. *Handbook of Cyanobacterial Monitoring and Cyanotoxin Analysis*. John Wiley & Sons.

Pyo, J., H. Duan, S. Baek, M. S. Kim, T. Jeon, Y. S. Kwon, H. Lee, and K. H. Cho. 2019. "A Convolutional Neural Network Regression for Quantifying Cyanobacteria Using Hyperspectral Imagery". *Remote Sensing of Environment* 233:111350.

Risco-Martín, J. L., S. Mittal, K. Henares, R. Cardenas, and P. Arroba. 2022. "xDEVS: A Toolkit for Interoperable Modeling and Simulation of Formal Discrete Event Systems". *Software: Practice and Experience*:1–42.

Schmale, D. G., A. P. Ault, W. Saad, D. T. Scott, and J. A. Westrick. 2019. "Perspectives on Harmful Algal Blooms (HABs) and the Cyberbiosecurity of Freshwater Systems". *Frontiers in Bioengineering and Biotechnology* 128.

Vincent, W. F. 2009. "Cyanobacteria". *Encyclopedia of Inland Waters* 3:226–232.

Wu, Y., J. Zhang, Z. Hou, Z. Tian, Z. Chu, and S. Wang. 2022. "Seasonal Dynamics of Algal Net Primary Production in Response to Phosphorus Input in a Mesotrophic Subtropical Plateau Lake, Southwestern China". *Water* 14(5).

Wymore, A. W. 2018. *Model-Based Systems Engineering*, Volume 3. CRC press.

Zeigler, B. P., A. Muzy, and E. Kofman. 2018. *Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations*. Academic press.

AUTHOR BIOGRAPHIES

SEGUNDO ESTEBAN SAN ROMÁN is an Associate Professor in the Department of Computer Architecture and Automation at Complutense University of Madrid (UCM). He received his Ph.D. in Physic Sciences from the UCM in 2002. His research is focused in practical aspect of modelling, simulation and control of dynamic systems. His email address is sesteban@ucm.es.

GIORDY A. ANDRADE AIMARA is a research support staff member in the Systems Engineering, Control, Automation and Robotics Research Group (ISCAR) at Complutense University of Madrid (UCM). He is Electronic Engineer and is currently

studying the Master of Systems and Control Engineering at the UCM-UNED universities. His research interests include simulation and control and optimisation of dynamic systems, deployment of electronic systems and internet of things. His email address is giordyan@ucm.es.

JOSÉ L. RISCO-MARTÍN received his Ph.D. from Universidad Complutense de Madrid (UCM), Spain, where he currently is Full Professor in the Department of Computer Architecture and Automation. His research interests focus on computer simulation and optimization, with emphasis on discrete event modeling and simulation, parallel and distributed Simulation, artificial intelligence in modeling and optimization, and feature engineering. His email address is jlrisco@ucm.es. His website is https://www.ucm.es/jlrisco.

JESÚS CHACÓN SOMBRÍA received his Ph.D degree in Control and Systems Engineering from the Universidad Nacional de Educacion a Distancia (UNED), Madrid, in 2014. He is an Assistant Professor in the Department of Computer Architecture and Automation. His research interests include simulation and control of event-based systems, communication protocols, and control education. His email address is jeschaco@ucm.es.

EVA BESADA-PORTAS received her Ph.D. in Computer Engineering from the Complutense University of Madrid (UCM), Madrid, Spain, in 2004. She is an Associate Professor in the Department of Computer Architecture and Automation. Her research interests are in modelling and simulation, heuristic optimisation, artificial intelligence and machine learning, unmanned vehicles, path planning and control, multi-sensor data fusion, and control education. Her email address is ebesada@ucm.es.