information

MDPI

Article

Modeling- and Simulation-Driven Methodology for the Deployment of an Inland Water Monitoring System

Giordy A. Andrade [1,*,†], Segundo Esteban [2,†], José L. Risco-Martín [2,†], Jesús Chacón [2,†] and Eva Besada-Portas [2,†]

1 System Engineering Control and Robotic Group, Universidad Complutense de Madrid, 28040 Madrid, Spain
2 Computer Architecture and Automatic Control Department, Universidad Complutense de Madrid, 28040 Madrid, Spain; sesteban@ucm.es (S.E.); jlrisco@ucm.es (J.L.R.-M.); jeschaco@ucm.es (J.C.); evabes@dacya.ucm.es (E.B.-P.)
* Correspondence: giordyan@ucm.es
† Current address: Facultad de Ciencias Físicas, Universidad Complutense de Madrid, Plaza de las Ciencias, 28040 Madrid, Spain.

Abstract: In response to the challenges introduced by global warming and increased eutrophication, this paper presents an innovative modeling and simulation (M&S)-driven model for developing an automated inland water monitoring system. This system is grounded in a layered Internet of Things (IoT) architecture and seamlessly integrates cloud, fog, and edge computing to enable sophisticated, real-time environmental surveillance and prediction of harmful algal and cyanobacterial blooms (HACBs). Utilizing autonomous boats as mobile data collection units within the edge layer, the system efficiently tracks algae and cyanobacteria proliferation and relays critical data upward through the architecture. These data feed into advanced inference models within the cloud layer, which inform predictive algorithms in the fog layer, orchestrating subsequent data-gathering missions. This paper also details a complete development environment that facilitates the system lifecycle from concept to deployment. The modular design is powered by Discrete Event System Specification (DEVS) and offers unparalleled adaptability, allowing developers to simulate, validate, and deploy modules incrementally and cutting across traditional developmental phases.

Keywords: Internet of Things; early warning system; harmful algal and cyanobacterial bloom; model-based system engineering; Discrete Event System Specification

check for updates

Citation: Andrade, G.A.; Esteban, S.; Risco-Martín, J.L.; Chacón, J.; Besada-Portas, E. Modeling- and Simulation-Driven Methodology for the Deployment of an Inland Water Monitoring System. *Information* **2024**, *15*, 267. https://doi.org/10.3390/info15050267

Academic Editor: Tieling Zhang

Received: 13 March 2024
Revised: 4 May 2024
Accepted: 5 May 2024
Published: 9 May 2024

1. Introduction and Related Work

Global warming and drought have rendered safe drinking water inaccessible for an estimated 4 billion individuals, constituting roughly two-thirds of the worldwide population [1]. Concurrently, population growth necessitates increased agricultural activity to satisfy burgeoning food demands, thereby exacerbating the depletion of water reserves. Previous contributions [2] provide an understanding of the changes in the water footprint for production and consumption for possible future scenarios by region.

Eutrophication is a principal contributor to the contamination of freshwater lakes and reservoirs. This phenomenon results from the proliferation of factors essential for photosynthesis, such as sunlight, carbon dioxide, and nutrients [3]. Natural eutrophication gradually transpires as aquatic ecosystems age and sediments accumulate. Nonetheless, the runoff from fertilizers, primarily due to intensive farming, hastens and intensifies this process.

Amidst the climate crisis and eutrophication of water bodies, there has been a noticeable uptick in the intensity and frequency of harmful algal and cyanobacterial blooms (HACBs) occurrences in recent decades. Notably, cyanobacterial blooms pose significant threats to public health and the environment. Cyanobacteria, a group of oxygen-producing bacteria, harness sunlight to metabolize carbon dioxide (CO_2) into organic matter [4]. These bacteria emerged over 3 billion years ago and catalyzed the oxygenation of Earth's atmosphere—a pivotal, transformative event [5]. Despite being commonly referred to as blue–green algae, they are not true algae; this term is reserved for eukaryotic photosynthetic organisms. Figure 1 displays a recent HACB in an inland water reservoir.

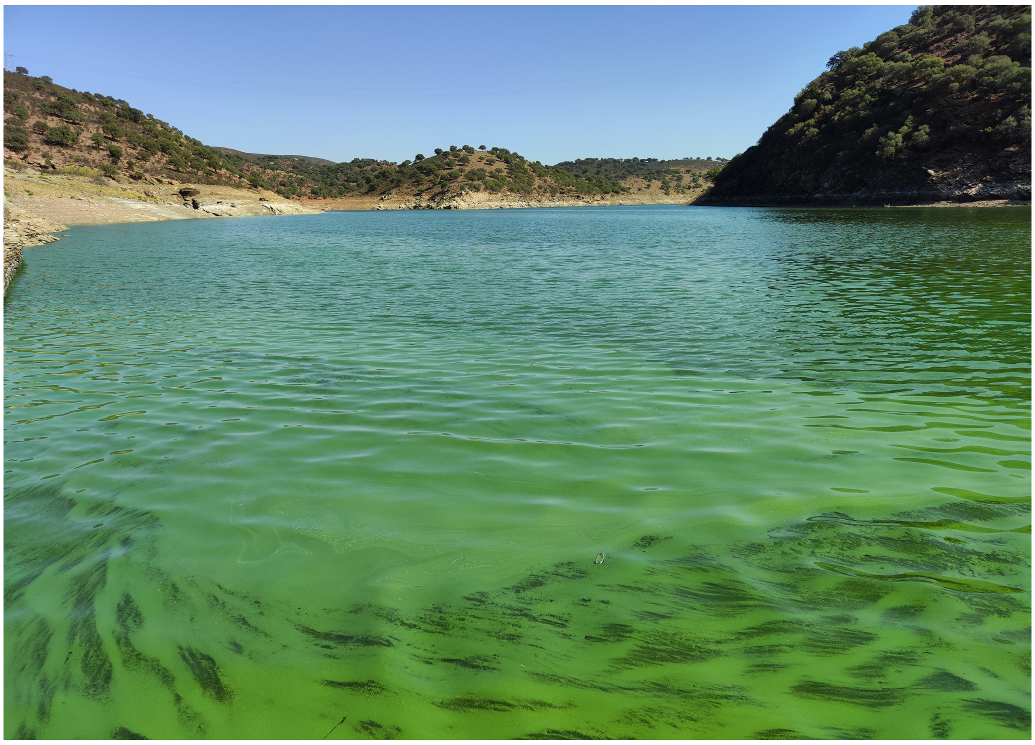


Figure 1. Photograph of a recent HACB in the Tajo and Tiétar Rivers confluence in Monfragüe Natural Park.

The peril extends beyond aquatic environments, as extracellular byproducts from HACBs have been detected in distant water and atmospheric samples and regions far from the coastline [6]. Addressing such incidents may necessitate substantial financial resources, mainly when environmental damage is extensive, thus constraining the available response options. This challenge has instigated a quest for systems adept at monitoring and mitigating environmental risks in aquatic settings.

Traditionally, since the 20th century, assessing water body health and HACBs prediction relied upon sample collection by specialists or automatic instruments positioned at set locations [7]. These samples would subsequently undergo laboratory analysis. Such methods demand considerable effort, time, and financial investment. Additionally, the procedural inefficiency and delayed analytical results hamper comprehensive study and impede the timely actions of entities responsible for water supplies and recreational water usage.

To safeguard the environment, developing new cost-effective early warning systems (EWSs) to track and forecast HACBs is imperative.

According to the United Nations Environment Programme, early warning is "the provision of timely and effective information, through identified institutions, that allows individuals exposed to hazard to take action to avoid or reduce their risk and prepare for effective response". Among the main features of an EWS are [8]: risk knowledge, where risk assessment is performed to prioritize the mitigation and prevention strategies; monitoring and prediction, composed of systems with monitoring and prediction capabilities to make timely risk estimates; information dissemination, for which communication systems are necessary to transmit reliable, synthetic, and simple warning messages to the authorities and the public; and response, composed of coordination and good governance, and appropriate action plans.

These EWSs must be capable of autonomous water analyses: processing informative data through modeling and simulation (M&S) techniques to predict the temporal and spatial dynamics of HACBs. For instance, machine learning models have proven effective for simulating HACBs in water systems [9]. It can be seen from previous works how obtaining knowledge from complex aquatic environment data sources is a challenge that requires extensive software engineering knowledge [10].

Recent advancements in environmental monitoring have led to the development of various predictive models and monitoring techniques aimed at addressing the challenges posed by HACBs (see [11]). However, despite these efforts, several open problems persist in the field. One of the primary issues is the lack of real-time data acquisition and processing capabilities that can provide immediate insights into the water quality and the presence of harmful algal blooms. Traditional methods often involve time-consuming sample collection and analysis, resulting in delays that can hinder prompt response and mitigation efforts. Furthermore, the integration of heterogeneous data sources, such as satellite imagery, in situ sensor networks, and meteorological data, remains a challenge. The effective fusion of these data streams is crucial for developing accurate and comprehensive predictive models. Additionally, there is a need for more sophisticated early warning systems that not only detect the presence of HACBs but also predict their movement and growth dynamics, enabling preemptive actions to be taken to protect public health and the environment [12].

The development of such a sophisticated EWS is a complex and multidisciplinary undertaking. This is why it requires collaboration between specialized teams focused on the parallel creation of the different subsystems of the system:

- HACB modeling: modeling of water bodies and inference of HACB biodynamics, with a segment of the team comprising biologists [12];
- Unmanned surface vehicles (USVs): guidance, navigation, and control (GNC) of USVs [13] and sensor instrumentation.
- Software and Internet of Things (IoT): software development and system simulation and integration [14].

Our solution addresses these challenges by introducing a novel M&S-driven EWS integrative model that leverages the power of IoT architecture [15] to facilitate real-time data acquisition and processing. By employing autonomous boats equipped with sensors, our system can gather high-resolution data on water quality parameters that are essential for the detection and prediction of HACBs. The integration of cloud, fog, and edge computing layers enables the efficient processing and fusion of diverse data sources, providing a comprehensive view of the aquatic environment. Our predictive algorithms, situated within the fog layer, are designed to offer timely predictions of algal bloom dynamics, thus addressing the critical need for advanced early warning capabilities. This article outlines a methodology for managing the intricate process of developing such an EWS. Modular and hierarchical formalism, vital for modeling and simulating such complex systems, is instrumental during development. Throughout this process, the Discrete Event System Specification (DEVS) formalism has been employed as a modeling technique that facilitates transitioning through the stages of development.

One of the main contributions of this paper is the detailed approach to subsystem verification and validation using the DEVS formalism. This methodology is particularly advantageous in the development of complex systems, such as the automated inland water monitoring system presented here. By leveraging DEVS, we can systematically replace virtual components with their real-world counterparts, enabling a gradual transformation from a virtual model to a fully operational real system. This approach not only saves costs by identifying and addressing potential issues early in the development process but also significantly reduces risks associated with deploying untested components in critical environments.

The ability to validate individual subsystems in isolation, and later in conjunction with other components, ensures a high level of confidence in the system's reliability and performance before it is fully deployed. This paper demonstrates the practical application of DEVS for achieving a structured and efficient transition from model to reality in the development of an early warning system for HACBs.

The subsequent content of this article is organized as follows: initially, the model-based development and DEVS formalism utilized for problem modeling is detailed; then, the description of the system is provided, followed by an exposition of the developmental stages: design, implementation, integration, verification, and validation; finally, the conclusions are shown.

2. Model-Based Development

Model-based system engineering (MBSE) is an approach that uses models to design, verify, and validate complex systems [16]. This method has become increasingly popular in systems engineering as it provides a visual and formal representation of systems: facilitating understanding and communication between the various stakeholders involved in developing and maintaining a system.

Unlike industrial projects, research projects usually do not have commercial solutions for some of the subsystems, and these subsystems will not be available until the final stages. Hence, it is necessary to advance in the development stages using simulated models of these subsystems. Therefore, with respect to the typical V-diagram used in engineering projects, in research projects, it will be necessary to verify and validate subsystems against other simulated subsystems that are based on models. Although software can be functionally verified on simulated subsystems, the validation will be performed on the real subsystem.

Figure 2 illustrates a hybrid V-model [17] with the software development stages of a research project. The V-model is called hybrid because it is not purely incremental: this allows agile iteration of designs and implementations based on the results obtained in verification and validation stages.

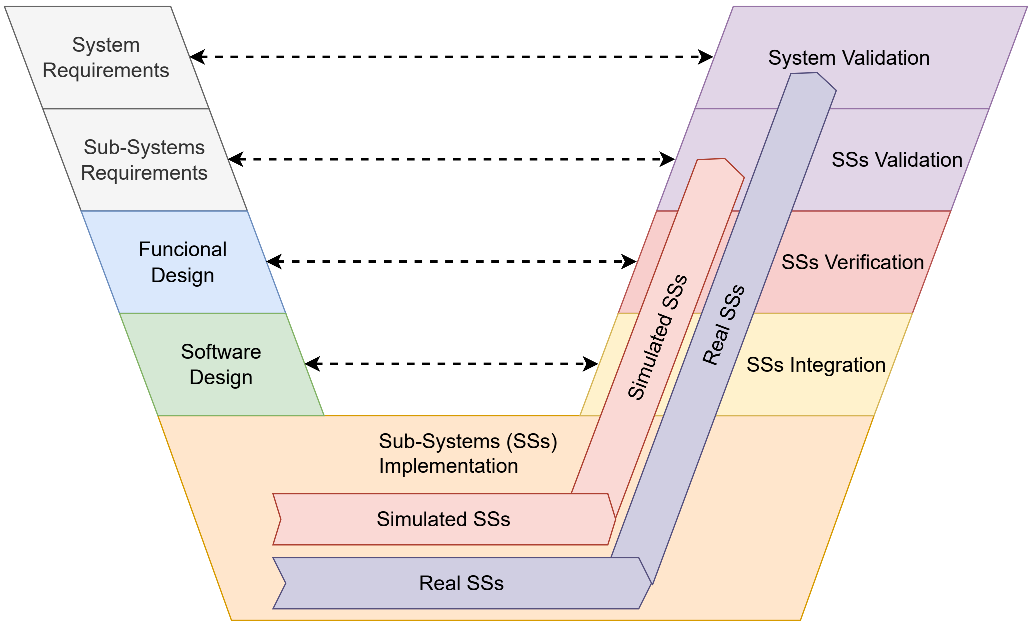


Figure 2. Research project development stages hybrid V-diagram. Real subsystems must be verified and validated versus simulated subsystems.

For model-based development, after defining the system's requirements and each subsystem, the team will design the functional models of the subsystems and then the software to merge them. Based on these designs, the functional models of the subsystems and the software logic will be implemented. In the integration stage, it must be ensured that the software correctly connects the different subsystems using their simulated models. The functional verification of subsystems is also performed on the simulated models, although the real subsystem can be used if available. On the other hand, in the subsystem validation stage, the real subsystem must be used, even if the rest of the subsystems are simulated. The final stage of system validation is where all the real subsystems should be used.

The modeling-to-deployment process presented in this paper requires advanced modeling and simulation technologies. These technologies provide a means to understand, analyze, and predict the behavior of these systems before they are implemented in the real world. One such technology is DEVS [18], which is a formalism widely used in modeling and simulation.

The complexity of the systems being dealt with requires using an M&S formalism. These systems often involve multiple interacting components, each with its own behaviors and interactions. Previous work has proposed methodologies based on model-driven engineering aimed at developing secure data management applications [19] or reducing the number of skills needed to manage a Big Data pipeline and support the automation of Big Data analytics [20].

Modeling and simulating such systems without a formalism can quickly become overwhelming and error-prone. In previous studies, Architecture Analysis & Design Language and DEVS are gradually used to develop combined architecture and design models [21]. In this regard, DEVS offers several benefits that make it suitable for developing complex systems. One of the key advantages is the ability to design the system's structure using coupled models and hierarchy. This allows the system to be broken down into manageable components and capture the interactions between them. By representing the system in this way, its behavior can be better understood, and informed decisions about its design and operation can be made. Another benefit is the separation of the simulator layer from the modeling layer. This separation allows for different simulation strategies, such as sequential, parallel, distributed, or even real-time simulation, without modifying the underlying model [22]. This flexibility is particularly valuable when dealing with complex systems that require different simulation approaches depending on the specific requirements or constraints.

Furthermore, DEVS provides a formal framework for modeling and simulation, which ensures consistency and rigor in the development process. The formalism allows for precise specification of system components, their behaviors, and their interactions, reducing ambiguity and facilitating communication among researchers and developers.

This section provides an overview of DEVS and discusses its key concepts and principles. Integration of real-time capabilities into DEVS has also been explored, which enables the simulation of systems that require timely and synchronized responses, being an internal process for deploying the real system.

2.1. DEVS Overview

Parallel DEVS is a formal methodology utilized for modeling discrete event systems, and it draws upon concepts from set theory [23]. The core components of parallel DEVS consist of atomic and coupled models, which can interact with other models through input (X) and output (Y) ports. Each atomic model maintains a state (S) that is associated with a time advance function denoted as ta, which governs the duration of the state.

Once the time allocated to the state has elapsed, it triggers an internal transition and activates an internal transition function ($\delta_{\text{int}} : S \to S$), leading to a modification to the local state ($\delta_{\text{int}}(s) = s'$). Simultaneously, the execution results of the model are disseminated through the output ports by means of an output function (λ).

Furthermore, external input events, which are received from other models, are gathered via the input ports. An external transition function ($\delta_{\text{ext}} : S \times e \times X \to S$) governs how to respond to these inputs, taking into account the current state (s), the time elapsed since the last event (e), and the input value (x) ($\delta_{\text{ext}}((s, e), x) = s'$). Parallel DEVS introduces a confluent function ($\delta_{\text{con}}((s, ta(s)), x) = s'$) that resolves conflicts between external and internal transitions.

The DEVS formalism serves as a powerful tool for specifying complex systems. By utilizing coupled models, atomic models, and their coupling relations, the structure of a complex system can be precisely defined. This allows for a clear representation of the system's components and their interactions, facilitating a comprehensive understanding of the system's behavior. Furthermore, the behavior of the system is defined through the atomic functions. These functions encapsulate the internal dynamics of each component and specify how the component responds to internal and external events. By defining the behavior of each atomic model, the overall behavior of the system can be accurately represented.

Once a system is described following the principles of DEVS theory, it can be readily implemented using one of the many DEVS simulation engines developed over the past few decades. Among these engines, xDEVS [24] is the one used in this work. It provides an exceptional solution for sequential, real-time, parallel, or distributed simulations.

2.2. Real-Time Integration

In the context of M&S, real-time integration refers to the capability of executing a simulation model in synchronization with real-world time [25]. This allows the simulation to respond to external events and interact with real devices in a timely and synchronized manner. In the case of the xDEVS simulation engine, real-time integration is achieved using the `RealTimeCoordinator` class.

The `RealTimeCoordinator` class is an extension of the `Coordinator` class in xDEVS and is specifically designed to enable real-time simulation. It introduces the concept of 'sigma waiting', where the coordinator effectively waits for the sigma values of the models until the output and transition functions must be executed. This ensures that the simulation progresses in accordance with real-world time, allowing for centralized synchronized responses.

In addition to the `RealTimeCoordinator` class, this paper introduces a new handler class called `RealTimeManager`. It is responsible for handling external events from real devices such as sensors or actuators. This enables the simultaneous handling of virtual and real components within the DEVS simulation model. From the simulation perspective, the hardware device is treated as an atomic model, allowing seamless integration and interaction. This enable incremental deployment of our complex EWS: replacing virtual elements with their real counterparts.

Figure 3 illustrates how the `RealTimeCoordinator` xDEVS coordinator class [26] has been slightly modified to incorporate the `RealTimeManager` handler.

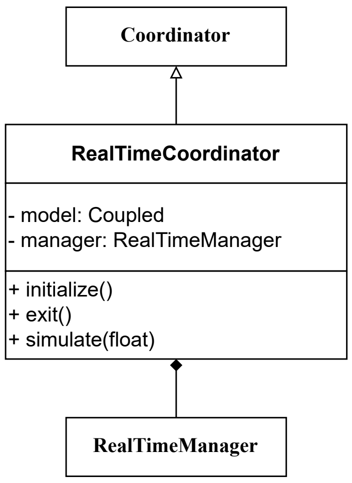


Figure 3. Unified Modeling Language (UML) diagram of the RealTimeCoordinator class.

The primary role of the `RealTimeManager` is to collect incoming messages and route them to the `RealTimeCoordinator` as well as to reschedule the corresponding external transition functions. The `RealTimeCoordinator` is responsible for coordinating both external events and internal DEVS model events. The implementation details of the input/output mechanism for handling real devices is provided in Section 8 of this paper.

3. Description of the System

This research aims to develop an automated EWS for monitoring HACBs. This system intends to integrate automatic data acquisition with parameterized model simulation. The approach is oriented towards a systems architecture, a more holistic and integrative model that includes the mathematical models mentioned above, and a modern model of the automated infrastructure of EWSs. Consistent with the IoT paradigm, the approach has been segmented into three distinct layers, which are depicted in Figure 4 and with each encompassing the various subsystems necessary for addressing the challenge.

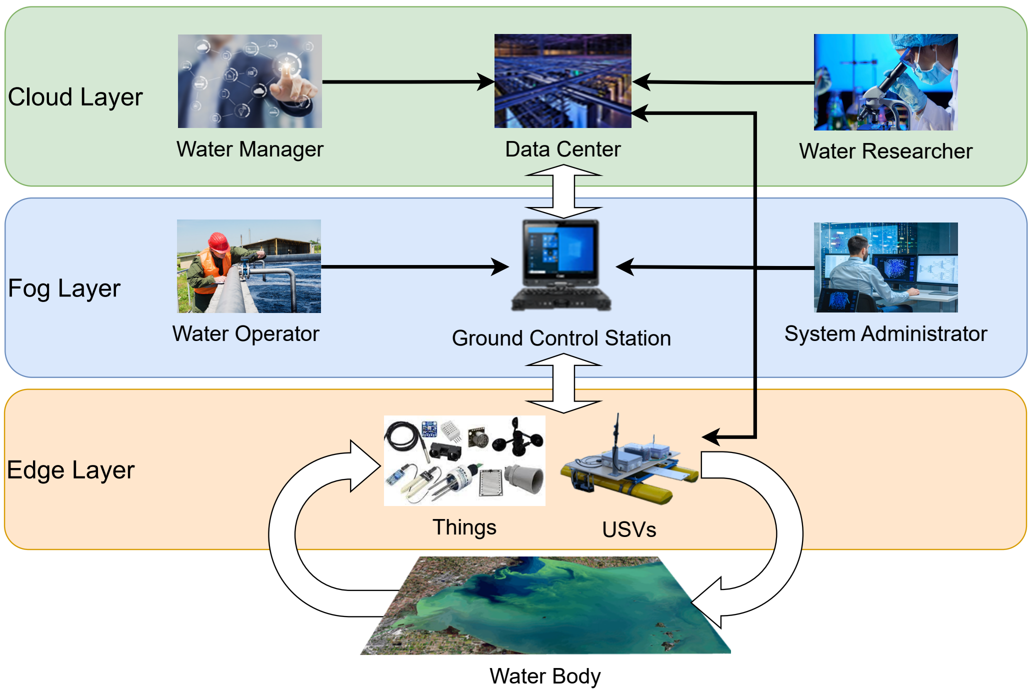


Figure 4. Conceptual design of the IoT layered structure.

The edge layer aggregates the system's physical components, such as sensors, weather stations, and the unmanned surface vehicles (USVs). These elements measure the conditions of the aquatic environment and are coordinated by the ground control station (GCS) located in the intermediary layer, which is known as the fog layer. The GCS employs an inference service to predict the emergence of HACBs and to direct the USV, ensuring that the onboard sensors target the appropriate regions. A water operator supervises and controls the system using the GCS.

Data are archived on servers within the final tier, which is denoted as the cloud layer and where reports and notifications are also produced for the water managers and researchers. Additionally, the inference and alert models are trained using the archived historical data in the cloud layer. Notably, the exchange of information, execution of control logic, and training loops among the layers must be realized in a closed-loop configuration. The system administrator will update the system software with the last trained models.

Before considering system deployment, it must be ensured that the control loops return stable outputs to avoid possible material or biological damage. Therefore, the inference models must be trained before deployment. In addition, after this training, it is necessary to verify the system's behavior in a simulated way.

4. Design Stage

The design phase begins with a basic coupled model that contains other models, atomic or coupled, to represent the functionality of each subsystem. Through an iterative process, the models and connections are refined until they represent the fundamental behavior of the subsystems. Figure 5 shows the initial distributed system model and information flows, where the subsystems that form it and the flow of information between them can already be appreciated.

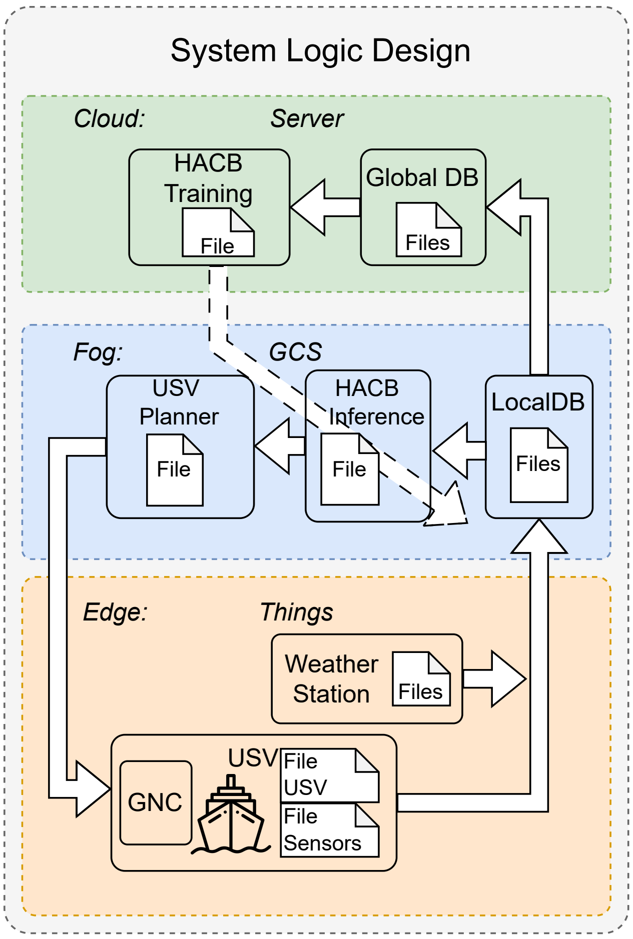


Figure 5. Distributed system model and information flows. Things in the edge layer capture information from the environment and feed it to the GCS, which resides in the fog layer. In this layer, the HACBs positions are inferred and the USV trajectory is planned. In parallel, all the information is uploaded to the cloud layer server to retrain the inference models. Subsequently, in a supervised way, the inference models are updated on the fog layer.

At this stage, the hardware and software that constitute the systems are not yet available, so input files are used to emulate the functionality of subsystems and events. The main objective of this model is to define the logic, timing, and message format between the atomic/coupled models.

In this design, the subsystems have been distributed in three layers. The edge layer includes two models: the ship with sensors and the weather station. The fog layer provides the behavior of the GCS, which employs three models: a local database, an inference model, and a trajectory planner. The cloud layer includes two models: a global database and an inference model trainer.

The logical behavior is simulated with files as follows. Initially, the inference model proposes a place and time to start data collection. Then, the planner sends the ship to that position. When the ship reaches the position, the ship's sensors take measurements, which are sent back to the GCS.

The GCS merges these measurements and those from the weather station, which generates signals periodically. The inference model can now use these measurements to close the control loop, inferring a new position and time to take measurements. In parallel, all this information is uploaded to the cloud layer server, where the inference model is trained again. When new inference parameters are available, they are also updated in the model used in the fog layer.

To clarify, the inference and its training work with different temporal granularity. A new position is inferred approximately every 30 min, while the model is only trained once a day (using the history received during the previous day, which will be available at the end of the day).

From this high-level design, design decisions are made iteratively. The structure of folders and files, the timing of messages, the type of messages, the separation between services and models, etc., have been refined. For example, through this process, it has been concluded that it is most advisable to generalize communications between subsystems and use a generic message class for the whole environment, as shown below.

```
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 potential of Python dictionaries, which are used in the `payload` field, allows all ports in the environment to share the same event class. Thanks to this, the development of web reports and graphical scenarios is substantially simplified.

5. Implementation Stage

Once an architecture comprising the models with their interconnections and the type of information to be exchanged has been defined, it is necessary to implement the functional behavior of these blocks. In cases where the USV and sensors are unavailable, emulation is utilized to simulate their interaction with the water body.

5.1. Simulated Body

A service called `SimBody` has been built to query the physical and biological state of the water body at a given time. This service internally uses a file previously generated with EFDC Explorer Modeling System (EEMS) [27]. Its use in Python is quite simple, as seen in the following code.

```
bodyfile: str = "./data/Washington-1m-2008-09_UGRID.nc"
simbody: SimBody = SimBody("SimWater", bodyfile)
```

This service will be used in the server to pre-train the inference service to generate acceptable predictions at the beginning of the simulation. Training these models before deploying them is necessary to avoid damaging the system, since they create the reference for the control loop. Subsequently, during the simulation or the system's actual operation, the inference model's training should be performed only with information from the lower layers.

From a computational point of view, EEMS simulations are pretty expensive. In addition, the files generated by EEMS are hefty and costly. Therefore, this simulator runs in the cloud layer on servers that can contain multiple simulations of various water bodies. To provide access to `SimBody` from the other layers, a web service called `RestBody` has been built. This service has been created using the FLASK library, and communications with the service are in JSON format. The following code shows the ease of use of this service, and it transparent to the user on the layer where it resides:

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

5.2. Things

Some previous studies show how to develop an environment model to predict reliability and availability when the actual hardware is inaccessible [28]. In the present research, the SimBody service is also employed by the simulated sensors, as shown below, to enhance their functionality and provide valuable data for the simulation.

```
sensor_info_n = SensorInfo(id=SensorEventId.NOX,
description="Nitrogen Sensor (mg/L)",
tinit=10, tmeasuring=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 code snippet above defines a nitrogen sensor model in Python. The `SensorInfo` class is used to specify parameters such as sensor ID, description, measuring time, maximum and minimum values, precision, noise bias, and noise sigma.

The `SimSensor` class then creates an instance of a DEVS atomic model named 'SimSenN' using the specified sensor information and a simulated body (`simbody`). As a result, it sets up the characteristics of a nitrogen sensor and creates a simulation object for it.

The logical behavior of the sensor is emulated by an atomic model that includes the state logic shown in Figure 6. The functional behavior of the sensor is provided by the `SimBody` service.

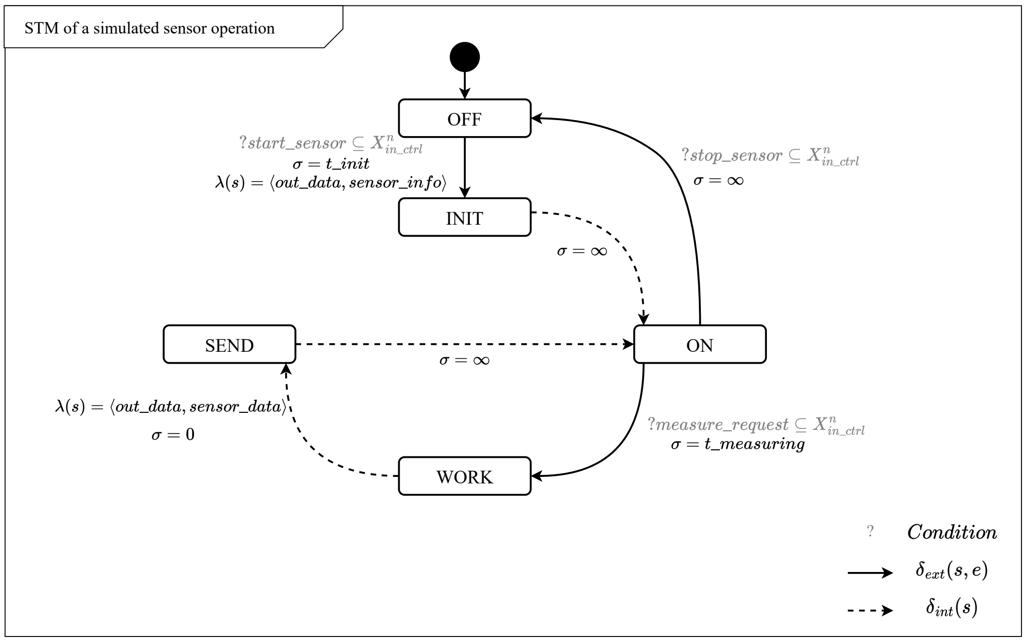

STM of a simulated sensor operation
OFF
INIT
ON
SEND
WORK
$?start\_sensor \subseteq X^n_{in\_ctrl}$
$\sigma = t\_init$
$\lambda(s) = \langle out\_data, sensor\_info \rangle$
$?stop\_sensor \subseteq X^n_{in\_ctrl}$
$\sigma = \infty$
$\sigma = \infty$
$\sigma = \infty$
$\lambda(s) = \langle out\_data, sensor\_data \rangle$
$\sigma = 0$
$?measure\_request \subseteq X^n_{in\_ctrl}$
$\sigma = t\_measuring$
? Condition
$\delta_{ext}(s, e)$
$\delta_{int}(s)$

Figure 6. STM of a simulated sensor operation.

The sensor logic includes the following states and transitions:

- OFF: The sensor is waiting for an activation control command.
- INIT: When the sensor is activated, it takes a time t_{init} to synchronize the time and to send back the sensor information.
- ON: The sensor is waiting for a measurement request or a stop control command.
- WORK: When a measurement is requested, the sensor takes a $t_{measuring}$ interval of time to obtain the measurement.
- SEND: The sensor sends the registered measurement.

One of the most complex Things to model are USVs because they have dynamic behavior that is subject to control actions and environmental perturbations. For this reason, they require a coupled model that, in addition to including simulated sensors, must include an atomic model describing the dynamics of the USV. This dynamic behavior can be modeled as a two-dimensional second-order differential equation, as in Equation (1).

$$\begin{aligned}
\frac{d^2 usv_{pos}}{dt^2} &= usv_{con}(t) + usv_{drg}(t) \\
usv_{con}(t) &= K_p \cdot (usv_{ref}(t) - usv_{pos}(t)) \\
usv_{drg}(t) &= K_{usv} \cdot \left(water_{vel}(t, usv_{pos}) - \frac{dusv_{pos}}{dt} \right) \\
&\textit{Subject to Control Saturation}
\end{aligned} \tag{1}$$

where the dynamics of the ship include control proportional to the position error $usv_{con}(t)$ and a drag perturbation by water currents $usv_{drg}(t)$. Thanks to this perturbation, the ship will follow the bloom almost without expending energy because it is dragged by the currents in the same direction as the bloom. The water velocity signal is obtained through the `SimBody` service. In addition, saturation of control signals is applied.

5.3. Inference Model

Initially, a straightforward inference model is implemented, Equation (2), which will be replaced by more complete models in the future. A two-dimensional second-order differential equation models the position of the bloom, and a first-order differential equation models the density of the bloom.

$$\begin{aligned}
\frac{d^2 blm_{pos}}{dt^2} &= K_{drg} \cdot \left(water_{vel}(t, blm_{pos}) - \frac{dblm_{pos}}{dt} \right) \\
\frac{dblm_{den}}{dt} &= K_{pho} \cdot sun(t) \cdot nox(t, blm_{pos}) + K_{oxi} \cdot dox(t, blm_{pos}) \cdot nox(t, blm_{pos}) \\
&\textit{Subject to Bloom Saturation}
\end{aligned} \tag{2}$$

where the first equation models the two-dimensional bloom position blm_{pos}, which is dragged by water currents, and the second equation calculates the bloom density blm_{den}. When solar radiation is present, density growth is proportional to nitrate photosynthesis $K_{pho}(sun(k) \cdot nox(k))$. During nights, if oxygen is present, growth is due to nitrate oxidation $K_{oxi} \cdot dox(k) \cdot nox(k)$. Real or simulated sensors should provide values for water velocity, solar radiation, and oxygen and nitrate levels. In simulation, they may be obtained using the `Simbody` service.

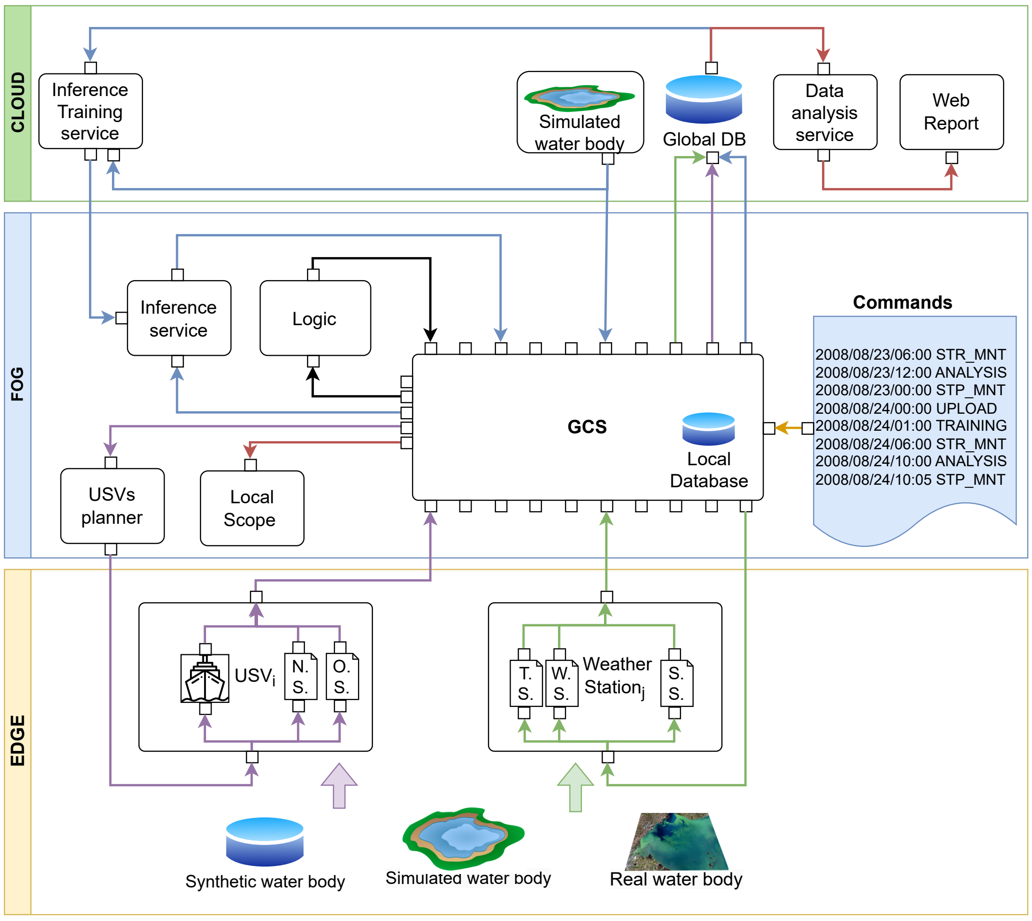
Communications have yet to be emulated in high detail at this development level. We have only emulated that the cloud database is updated at the end of the day. When new data become available, the inference model is trained. Only if the supervisor considers that the new training provides significantly better results are the model parameters are updated. In the current version, least squares training is performed, which seeks to reduce the error between the predictions made by the inference model and those provided by the sensors, as in Equation (3). In the near future, other types of models will be available, so the supervisor will be able to evaluate them comparatively and use the one that offers the best results.

$$\begin{aligned}
\begin{bmatrix} \hat{K}_{pho} \\ \hat{K}_{oxi} \end{bmatrix} &= \underset{K_{pho}, K_{oxi}}{\operatorname{argmin}} \sum_{t=t_0}^{t_{end}} \Big(EEMS_{den}(t) - blm_{den}(t, K_{pho}, K_{oxi}) \Big)^2 \\
\hat{K}_{drg} &= \underset{K_{drg}}{\operatorname{argmin}} \sum_{t=t_0}^{t_{end}} \Big(EEMS_{pos}(t) - blm_{pos}(t, K_{drg}) \Big)^2
\end{aligned} \tag{3}$$

where $EEMS_{den}(t)$ is the maximum density found in the map generated by EEMS for time t, and $EEMS_{pos}(t)$ is the two-dimensional position where that maximum density is found. These values can be pre-calculated in the cloud layer based on the EEMS density map. Training generates new constants for the inference model, which are sent to the fog layer to be used in future inferences.

6. Integration Stage

The integration stage of our development process starts with the initial model, which was introduced during the design phase and which matures into a sophisticated coupled DEVS model as depicted in Figure 7. Other, more complete models may include warning and forecasting services, such as those shown in [12].



```
2008/08/23/06:00 STR_MNT
2008/08/23/12:00 ANALYSIS
2008/08/23/00:00 STP_MNT
2008/08/24/00:00 UPLOAD
2008/08/24/01:00 TRAINING
2008/08/24/06:00 STR_MNT
2008/08/24/10:00 ANALYSIS
2008/08/24/10:05 STP_MNT
```

Figure 7. Coupled model of the architecture for bloom monitoring using an inference service.

This model now encapsulates all the necessary subsystems to effectively monitor an HACB and employs a trainable inference service that guides an instrumented USV. The evolution of the model through the implementation stage has been instrumental for refining the interactions and functionalities of the subsystems, ensuring a seamless integration within the overall architecture.

In this model, the USV dynamics and control are simulated according to Equation (1), the sensor measurements are emulated using the `SimBody` service, the inference service is used in the fog layer according to Equation (2), and it is trained on the cloud layer according to Equation (3). In this simple case, the planner sends the inferred blm_{pos} to the USV as a reference. The USV will have to finish reaching the marked reference point: it will have to adjust its trajectory as its dynamics are altered by the influence of water currents and wind speed. In addition, an atomic model called `Commands` has been included that emulates the water operator manipulating the GCS. This has significant added value because it allows the system to give orders during the simulation.

The integration stage is a critical phase in developing complex systems, where the individual components, which were previously developed and tested in isolation, are brought together to form a cohesive and functioning whole. Adopting the DEVS formalism throughout the software design and implementation stages offers a structured approach to this process. Firstly, it ensures that the integration of subsystems is not an afterthought but a consideration embedded in the development process from the beginning. Secondly, it allows for a more modular design, where subsystems can be developed and tested independently, reducing the complexity of the integration stage. Additionally, the DEVS formalism facilitates the transition of models from the development stage to the integration, verification, and validation stages. Models developed under this formalism can quickly evolve and adapt as they move through these stages with the confidence that their integration has been considered from the outset. This is particularly beneficial in research projects where some subsystems may not have commercial solutions available until later stages, necessitating the use of simulated models for progression.

7. Verification Stage

This coupled model already allows us to verify the functional behavior of the inference model and its training in a simulated way. Figure 8 shows an enriched representation of the evolution of the tracking and monitoring of an algal bloom by the USV.

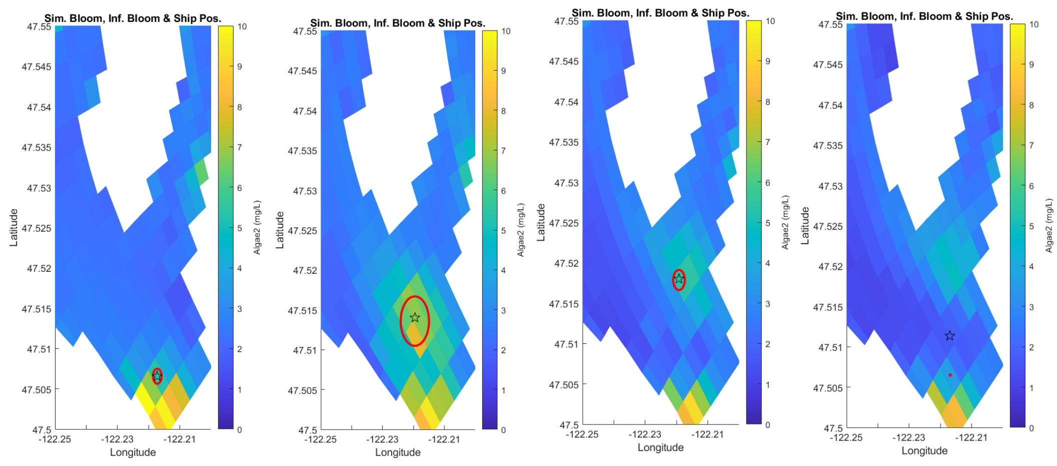


Figure 8. Augmented representation during a bloom monitoring simulation. The plots show the EEMS map of the algal density in the surface layer, the position of the USV (black star), and the position and density of the inferred bloom (red circle). Each plot shows a different moment: the bloom hatching at 7:30 AM (top-left), the movement at 15:30 (top-right), the dispersion at 22:30 (bottom-left), and when the USV returns to the bloom's incubator at 2:00 AM (red dot in right subfigure). Available at: https://archive.org/details/habsim-20220920, (accessed on 4 May 2024).

This figure shows the position indicated by the inference model and the position reached by the USV: all are enriched with the algal density map provided by EEMS. By analyzing the time evolution of the simulation and its visual representation, it can be verified that the inference model is working correctly.

A quantitative functional verification of the bloom inference model and its monitoring is shown in Figure 9. The figure shows the behavior of the system over several days. It can be seen that the system correctly detects the occurrence of blooms and predicts their density with acceptable accuracy. In addition, it also generates the bloom's displacement, which is used by the USV to take bloom density measurements. It can be seen that at the beginning of a new day, the model is initialized to the initial position instantaneously, but the USV must return during the night.

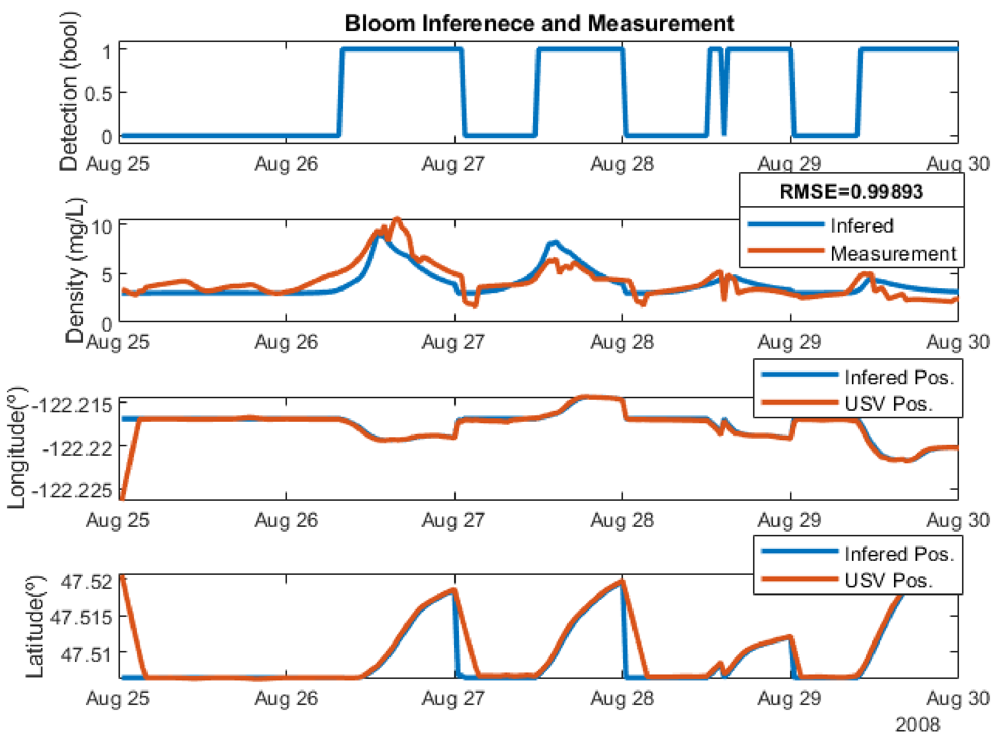


Figure 9. Functional verification of the inference model over several days. In the first plot, the model predicts the existence of a possible bloom. The second plot shows the density inferred by the model, the density measured by the USV, and the mean square error. The third and fourth plots show the longitudes and latitudes inferred by the bloom model and those attained by the USV.

8. Validation Stage

The next stage consists of transforming the simulator into real-time software that allows the deployment and validation of real subsystems. This must be done progressively, i.e., subsystem by subsystem. To do this, part of the software must be re-coded to run on the embedded hardware, and another part can evolve from the simulation version to a real-time version, taking advantage of the DEVS formalism. DEVS allows the model to work in both simulated- and real-time and also in accelerated real-time, thus speeding up the validation process.

Previous studies have shown possible robust management strategies for IoT sensors deployed in real environments and have shown the coordination between indoor and outdoor sensors to extend real-time monitoring coverage [29,30]. To deploy and validate a subsystem, it must communicate with the rest of the subsystems. Therefore, in such a complex system, a mixed deployment is necessary, where one subsystem runs on real hardware and the others are simulated.

All subsystems have already been modeled and integrated into the simulator. Most of them have already been developed and are in the verification stage. A real GPS at the edge layer and the `WebReports` service at the cloud layer have been validated.

8.1. GPS Validation

To carry out the validation phase of some subsystems, real Things must be integrated into real-time simulations, for which a common communication interface must be incorporated.

As explained in Section 2.2, the motivation behind I/O handlers is to be able to adapt external events as input messages and send output messages outside the simulator. The DEVS simulator was not originally planned to handle external events. This can be implemented in two ways: by event polling or by event injection.

- Things integration by polling: The simulator requests information from the Things through a communications protocol. First, the necessary connections must be opened, and the parameters must be configured. Once the simulation is launched, the processor is responsible for making a request to the device being polled, where both the input data (captured by *read_data* function) and the device status can be queried. Waiting for a response can be active if the execution is blocked until a response is received, or it can be passive if the simulation continues to run regardless of whether a response is received or not. The processor is in charge of checking the presence of accumulated data in a specific virtual device every so often. Subsequently, information processing is performed, which may involve data analysis or decision-making based on the device status, etc. After processing the response, it generally waits for a certain period before sending the next request. This has the advantage of avoiding overloading the devices with recurring requests. This method is widely used in HTTP requests when working with web services or APIs [31].
- Things integration by event injection: In this method, an `Injector` is added to the simulator (external to the DEVS formalism), which allows external events to be incorporated in a soft real-time simulator [32]. External events occur outside the normal simulation flow, such as a hardware device sending a signal to indicate that it needs attention. Now, the information from the sensor is received by the `Injector`, which feeds an event into the simulator by the in_{rt_event} port. The simulator processes the event as soon as possible. Among the benefits of employing this type of integration is the speed of response, as it benefits the synchronization between the sensor time and the simulator clock. In this way, the data obtained by the sensor are directly reflected in the data that the simulator works with.

Figure 10 shows the two integration schemes for a GPS sensor. In these examples, it is possible to appreciate the connection of the external device with the virtual sensor of the simulator and the way to communicate with the GCS.

Although both methods work correctly with the designed system, we decided to work with the integration of the GPS sensor by injection because these data are externally generated events.

For the case of the sensor integrated by polling, the access to the data provided by the real sensor maintains the state space diagram design for simulated sensors shown above in Figure 6. The virtual sensor starts from the OFF state and completes its initialization by notifying the simulator that it is a real sensor and specifying its data types and communication protocol, and it transits to the ON state. If the simulator sends a request for data on the *in_ctrl* port, the sensor transits to a real WORK state, where the real sensor reads data from the real world rather than reading from the `simbody`. After the read time interval, the data are available and the virtual sensor transits to the SEND state to provide the simulator with an event with the information. Finally, the sensor returns to the ON state to wait for a new request from the simulator.

The case of the integrated sensor with event injection shown in Figure 11 is very similar to the simulated sensor diagram in the initial states. However, the sensor model remains in the ON state until it receives the external sensor data injection over the in_{rt_event} port, which causes an external state transition to the SEND state and sending of the sensor data to the simulator. In this way, the sending of sensor data to the simulator only depends on the arrival of an external interrupt with data from the external sensor to the simulator. For this reason, the WORK state and internal state transitions executed with a sigma period are dispensed with. Once the data injection is complete, the sensor returns to the ON state to continue normal operation. In this case, the simulator can also place the sensor in the OFF state as required.

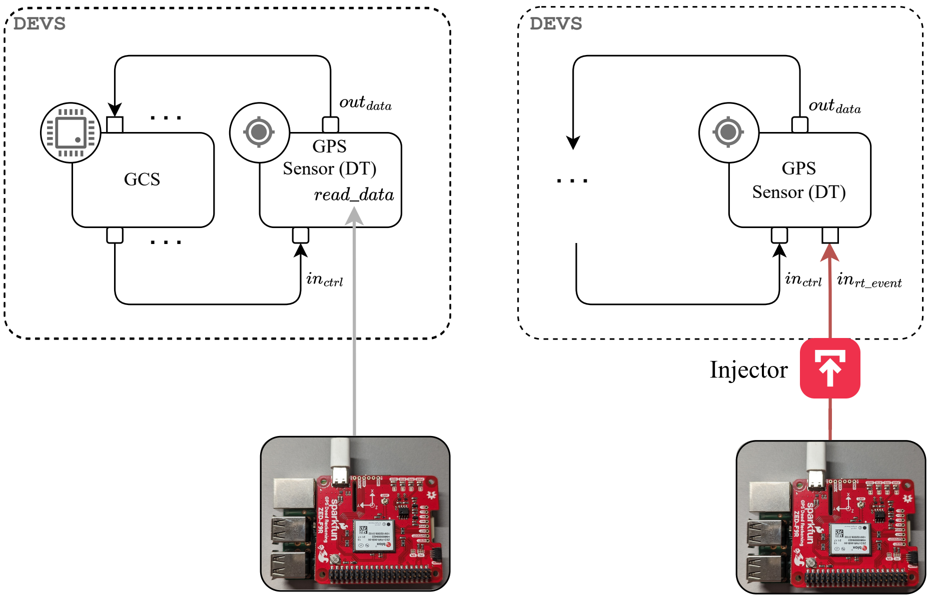


Figure 10. Integration of Things by polling (**left**) and integration of Things by event injection (**right**) in the simulator.

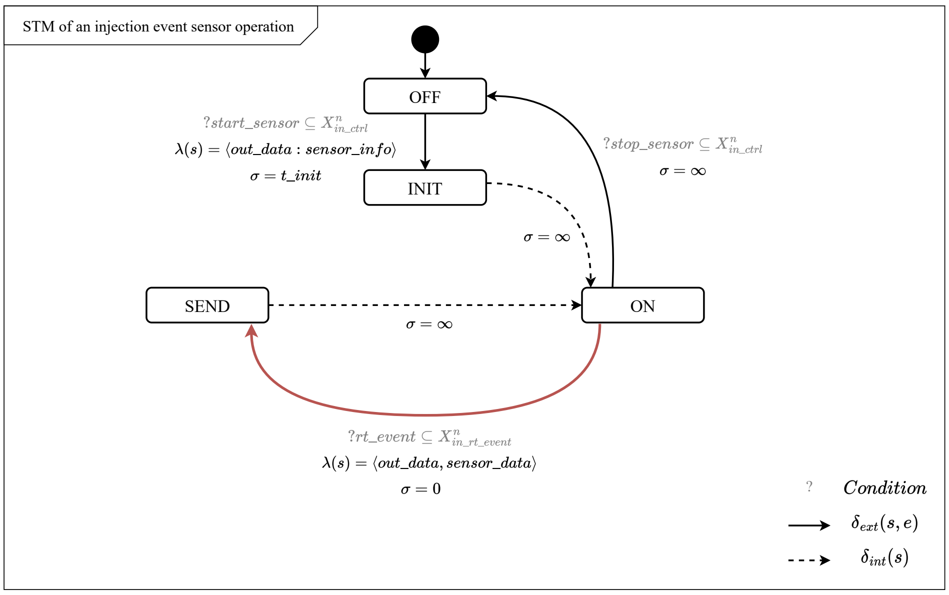

STM of an injection event sensor operation
OFF
INIT
ON
SEND
$?start\_sensor \subseteq X^n_{in\_ctrl}$
$\lambda(s) = \langle out\_data : sensor\_info \rangle$
$\sigma = t\_init$
$?stop\_sensor \subseteq X^n_{in\_ctrl}$
$\sigma = \infty$
$\sigma = \infty$
$\sigma = \infty$
$?rt\_event \subseteq X^n_{in\_rt\_event}$
$\lambda(s) = \langle out\_data, sensor\_data \rangle$
$\sigma = 0$
? Condition
$\delta_{ext}(s, e)$
$\delta_{int}(s)$

Figure 11. SysML state machine diagram of injection event sensor operation. The diagram forms part of a use case of the state machine diagram shown for the case of a sensor. Unlike this one, sensors implemented by event injection do not have a WORK state since their performance depends only on external events.

A commercial GPS sensor has been used for the validation stage, and a version of its drivers has been executed in real-time. The messages delivered by the GPS sensor follow the protocol provided by the National Marine Electronics Association (NMEA) [33]. Thus, the messages provided by the GPS are constantly collected by the `Injector` in a parallel thread containing a Flask server. In this case, only the message containing the information related to the position, the timestamp, the error, and the signal quality is of interest to be collected. The driver discards messages that do not correspond to these parameters.

Once the desired information has been obtained from the GPS, the `Injector` adds the measurement values into the simulator as an external event. A log of the software driver is provided below.

```
...
```

```
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
...
```

The validation of the GPS sensor shows an anomaly. A 2 h and 1 s lag exists between the timestamp of the simulator and the GPS timestamp. The 2 h lag is because the computer clock is set to local time, and the 1 s delay is because UTC corrects 1 s every five years to compensate for slowing of the earth's rotation.

8.2. WebReport Validation

This service resides in the cloud layer, and it is responsible for receiving data from the other layers, processing them, and displaying them on user request. The cloud layer includes servers and data centers that can store large amounts of data and perform data processing and analysis. Data analytics and machine learning, which require high computational power, can be performed on these servers.

As in the coupled fog model, the cloud model can run different services but is highly scaled to manage one or several water bodies. These services include running extensive data analysis, with all data stored in the central database, or running training services to update the inference models for the models coupled to the fog layer. In any case, these actions are always triggered by the arrival of new data files from the lower layers. For example, services have been implemented in the cloud layer to inform the water manager of alerts or reports generated throughout the simulation day.

No specific atomic models have been included for executing services because they are always processes installed in Docker containers, i.e., they have a distributed architecture. It is unnecessary to encapsulate such atomic models in DEVS models; in other words, the cloud layer is seen as a centralized entity.

Adaptability is guaranteed as the framework has been designed with scalability in mind, which allows users to outsource and scale up critical services in case of bottlenecks. For example, resource-intensive atomic models, such as those designed for training services or data analysis, can be easily simulated in parallel or distributed computing architectures.

Figure 12 shows the connection structure of the cloud layer, including its components. It can be seen that the database aggregates the information delivered by the fog layer and Things. In this case, the data are stored in directories with files hierarchically organized by the body of water in which the monitoring is performed, the date, and the type of signal recorded. The benefit of using this type of system is that it is possible to store data from different Things regardless of whether they are actual or simulated subsystems.

The superior subsystems, such as the web report or the other services, use the information provided by the database and can identify the nature of each signal by the type of file. However, the services work the same way for each case, even if real-time simulations are used. In this case, the files have been generated by simulation, but the service's validation is also valid for actual data files.

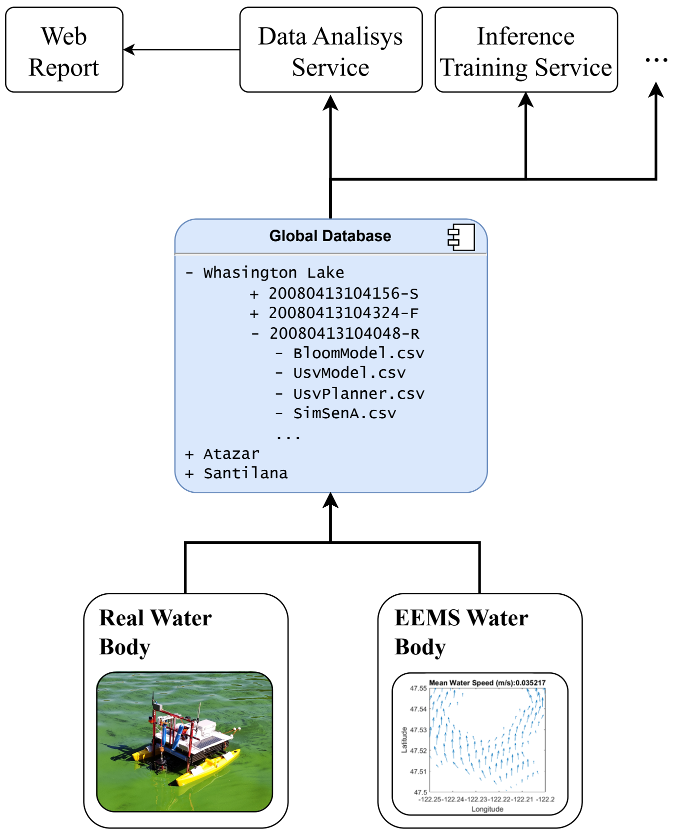


Figure 12. Connection structure of cloud layer with the simulator through the database.

Early warning systems are often accompanied by a web interface or application, which offers different services and has the potential to create a direct communication channel between citizens, researchers, and decision-makers [34]. There are countless applications for EWS monitoring, be it fire prevention [35], weather monitoring [36], geological hazards [37], biological hazards [38], etc. Among their most common features are alert generation, signal monitoring, signal analysis, and event forecasting. This type of web interface demonstrates the capabilities of the system to be able to integrate the collection, visualization, and analysis of sensor data on a single platform [39].

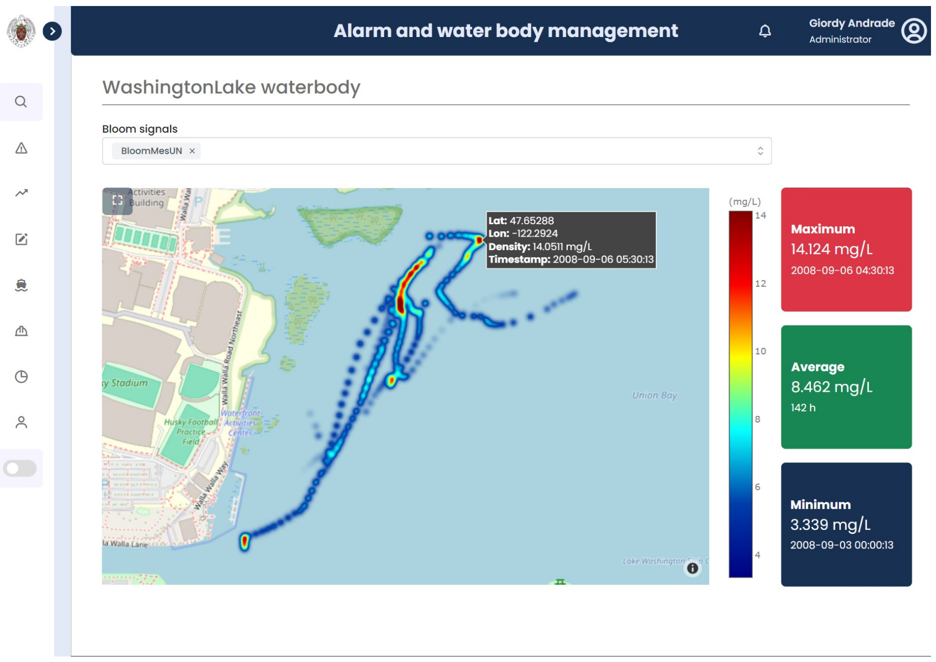
An advanced web application has been designed to monitor the HACBs in different water bodies. For this purpose, the free and open-source tool Django is used to develop an application efficiently and with a user-friendly interface that provides highly relevant information for the different user roles [40]. The interface has several views aimed at offering different services to users: the first of these views is 'Alarm and Water body management', which provides the water manager with visual information on the recording of algal bloom measurements and alarms generated in a water body. It is possible to delimit the regions of the map where the algae level alert service is performed and to mark the limit values. This information must be declared when initializing the simulator.

The 'USV's Monitoring' view allows the user to keep track of the planning and trajectories of the USVs used for the monitoring of the HACBs. The behavior of the ship(s) can be observed dynamically over the recorded days. Another of the views incorporated in the web interface is the 'Forecast' view, which is used to obtain estimates of the behavior of algal blooms based on historical data.

Finally, the last implemented view is the 'Signal Analysis' view, which is designed to provide managers with a detailed examination of the signals captured by the USV sensors. This functionality allows pattern analysis, identification of possible events, and evaluation of the quality of actions taken by authorities. The data obtained are presented graphically and interactively for more efficient interpretation. A view is currently being developed for an 'editor', which allows the user to configure the simulation scenarios and obtain data in regions of high interest or under certain specified conditions.

One of the views provided by the web service can be seen in Figure 13. It is the page destined for the water manager. This page shows a record of the different signals captured by the USV sensors for each of the available water bodies. In particular, the figure shows the measured and predicted algal density signal for the Lake Washington water body between 1 and 8 September 2008. The reactive interface allows filtering the signals by date and displaying the maximum, minimum, and average values and the instants they occurred. This screen lets us obtain information on the values collected by the ship's sensors found in a specified body of water along its course. A list of alarms is automatically generated according to the maximum values of the selected detection parameters (algal, nitrate, and oxygen levels).

The services react to the user in a reactive way according to the parameters specified by the manager. In each view, the user can select the water body, the signals to be represented, and the date range over which the data are found. In this way, the cloud layer can support authorities when they are making high-level decisions based on the information provided on the different water bodies.



Lat: 47.65288
Lon: -122.2924
Density: 14.0511 mg/L
Timestamp: 2008-09-06 05:30:13

Maximum
14.124 mg/L
2008-09-06 04:30:13

Average
8.462 mg/L
142 h

Minimum
3.339 mg/L
2008-09-03 00:00:13

Figure 13. Web view of 'Alarm and Water body management' for the water manager (map of the water body Washington Lake showing the algal density signal from 1 to 6 September 2008). Available at: http://pc-iscar.dacya.ucm.es:8080/water_management/, (accessed on 4 May 2024).

9. Conclusions

The presence of HACBs in aquatic ecosystems such as marshes, swamps, lakes, and rivers poses risks for recreational use and human consumption, adversely affects ecosystems, and directly impacts public health. Traditional efforts to detect these blooms have relied on reactive measures and isolated actions such as manual sample collection, automatic sampling at fixed locations, or the use of classical predictive models. However, new EWSs are being developed to provide proactive, contemporary, and automated monitoring systems.

In this project, we propose the design and development of an advanced EWS. Due to the system's complexity, we have undertaken a robust implementation based on model-based system engineering (MBSE) principles and driven by the DEVS approach. This methodology enables the progressive design, development, validation, deployment, and verification of a complex system such as the one we are focusing on.

During the development of the various subsystems, it is crucial to simulate them to ensure distributed validation and deployment. It is imperative to fully stabilize the control loops before deployment using a system simulator to train the inference models.

For the design of the framework architecture, we have stratified the system in layers according to the IoT paradigm: cloud, fog, and edge layers. The system was then developed in a simulated environment, allowing for validation of subsystems. This was followed by an incremental deployment process to progressively validate subsystems.

The research presented in this paper has successfully demonstrated the application of a modeling- and simulation-driven methodology for the development of an inland water monitoring system. The use of the DEVS formalism has provided a structured and systematic approach to the development process and enabled the simulation, validation, and incremental deployment of various subsystems. The layered IoT architecture has facilitated the integration of autonomous data collection units, predictive algorithms, and inference models, resulting in a robust and adaptable EWS for HACB surveillance.

The methodology outlined in this paper has significant implications for environmental monitoring and management. By leveraging the capabilities of DEVS modeling and real-time simulation, the developed system can provide timely and accurate predictions of HACB occurrences, thereby enhancing the ability of water managers and authorities to respond effectively to environmental threats. The modular design and the integration of real and simulated components offer a flexible and cost-effective solution for deploying complex monitoring systems in diverse aquatic environments.

Future work will focus on refining the inference models, improving the real-time data processing capabilities, and expanding the system's functionality to cover a broader range of environmental parameters. The ultimate goal is to establish a comprehensive monitoring framework that can be adapted to various ecological contexts and contribute to the global effort to combat the adverse effects of climate change on water resources.

Author Contributions: Conceptualization, E.B.-P., J.C., J.L.R.-M., G.A.A. and S.E.; methodology, J.L.R.-M., J.C. and S.E.; software, S.E., J.L.R.-M., J.C. and G.A.A.; validation, J.L.R.-M. and S.E.; formal analysis, G.A.A., S.E. and J.L.R.-M.; investigation, G.A.A. and S.E.; writing—original draft preparation, G.A.A. and S.E.; writing—review and editing, J.L.R.-M.; visualization, G.A.A.; funding acquisition, J.C., J.L.R.-M. and E.B.-P. All authors have read and agreed to the published version of the manuscript.

Funding: 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.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

Glossary

DEVS	Discrete Event System Specification
EEMS	EFDC Explorer Modeling System
EWS	early warning system
STM	state machine diagram
GCS	ground control station
HACB	harmful algal and cyanobacterial bloom
IoT	Internet of Things
M&S	modeling and simulation
MBSE	model-based system engineering
NMEA	National Marine Electronics Association
UML	Unified Modeling Language
USV	unmanned surface vehicle
GNC	guidance, navigation, and control

References

1. Mekonnen, M.M.; Hoekstra, A.Y. Four billion people facing severe water scarcity. *Sci. Adv.* **2016**, *2*, e1500323. [CrossRef] [PubMed]
2. Ercin, A.E.; Hoekstra, A.Y. Water footprint scenarios for 2050: A global analysis. *Environ. Int.* **2014**, *64*, 71–82. [CrossRef] [PubMed]
3. Chislock, M.F.; Doster, E.; Zitomer, R.A.; Wilson, A.E. Eutrophication: Causes, consequences, and controls in aquatic ecosystems. *Nat. Educ. Knowl.* **2013**, *4*, 10.
4. Huisman, J.; Codd, G.A.; Paerl, H.W.; Ibelings, B.W.; Verspagen, J.M.; Visser, P.M. Cyanobacterial blooms. *Nat. Rev. Microbiol.* **2018**, *16*, 471–483. [CrossRef] [PubMed]
5. Schirrmeister, B.E.; Gugger, M.; Donoghue, P.C. Cyanobacteria and the Great Oxidation Event: Evidence from genes and fossils. *Palaeontology* **2015**, *58*, 769–785. [CrossRef] [PubMed]
6. Schmale, D.G., III; Ault, A.P.; Saad, W.; Scott, D.T.; Westrick, J.A. Perspectives on harmful algal blooms (HABs) and the cyberbiosecurity of freshwater systems. *Front. Bioeng. Biotechnol.* **2019**, *7*, 128. [CrossRef]
7. Miller, T.R.; Tarpey, W.; Nuese, J.; Smith, M. Real-Time Monitoring of Cyanobacterial Harmful Algal Blooms with the Panther Buoy. *ACS EST Water* **2022**, *2*, 1099–1110. [CrossRef]
8. Grasso, V.F.; Singh, A. *Early Warning Systems: State-of-Art Analysis and Future Directions*; Draft Report UNEP; UNEP: Nairobi, Kenya, 2011; Volume 1.
9. Baek, S.S.; Pyo, J.; Kwon, Y.S.; Chun, S.J.; Baek, S.H.; Ahn, C.Y.; Oh, H.M.; Kim, Y.O.; Cho, K.H. Deep learning for simulating harmful algal blooms using ocean numerical model. *Front. Mar. Sci.* **2021**, *8*, 729954. [CrossRef]
10. Negreiros, B.; Schwindt, S.; Scolari, F.; Barros, R.; Galdos, A.A.; Noack, M.; Haun, S.; Wieprecht, S. A database application framework toward data-driven vertical connectivity analysis of rivers. *Environ. Model. Softw.* **2024**, *172*, 105916. [CrossRef]
11. Herguedas-Pinedo, B.; Risco-Martín, J.L.; Esteban, S.; López-Orozco, J.A.; Besada-Portas, E. Predictive Modeling and Simulation System for the Management of Harmful Cyanobacteria Blooms. In Proceedings of the 2023 Annual Modeling and Simulation Conference (ANNSIM), Hamilton, ON, Canada, 23–26 May 2023; pp. 122–133.
12. Risco-Martín, J.L.; Esteban, S.; Chacón, J.; Carazo-Barbero, G.; Besada-Portas, E.; López-Orozco, J.A. Simulation-driven engineering for the management of harmful algal and cyanobacterial blooms. *Simulation* **2023**, *99*, 1041–1055. [CrossRef]
13. Nađ, Đ.; Mišković, N.; Mandić, F. Navigation, guidance and control of an overactuated marine surface vehicle. *Annu. Rev. Control* **2015**, *40*, 172–181. [CrossRef]
14. Mittal, S.; Tolk, A.; Pyles, A.; Balen, N.V.; Bergollo, K. Digital Twin Modeling, Co-Simulation and Cyber Use-Case Inclusion Methodology for IOT Systems. In Proceedings of the 2019 Winter Simulation Conference (WSC), National Harbor, MD, USA, 8–11 December 2019; pp. 2653–2664. [CrossRef]
15. Lakhwani, K.; Gianey, H.K.; Wireko, J.K.; Hiran, K.K. *Internet of Things (IoT): Principles, Paradigms and Applications of IoT*; Bpb Publications: New Delhi, India, 2020.
16. Madni, A.M.; Sievers, M. Model-based systems engineering: Motivation, current status, and research opportunities. *Syst. Eng.* **2018**, *21*, 172–190. [CrossRef]
17. Naeem, M.R.; Zhu, W.; Memon, A.A.; Khalid, A. Using V-Model methodology, UML process-based risk assessment of software and visualization. In Proceedings of the 2014 International Conference on Cloud Computing and Internet of Things, Changchun, China, 13–14 December 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 197–202.
18. Zeigler, B.P.; Muzy, A.; Kofman, E. *Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations*; Academic Press: Cambridge, MA, USA, 2018.
19. Somogyi, F.A.; Asztalos, M. Systematic review of matching techniques used in model-driven methodologies. *Softw. Syst. Model.* **2020**, *19*, 693–720. [CrossRef]

20. Ardagna, C.A.; Bellandi, V.; Ceravolo, P.; Damiani, E.; Bezzi, M.; Hebert, C. A model-driven methodology for big data analytics-as-a-service. In Proceedings of the 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, USA, 25–30 June 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 105–112.
21. Ahmad, E.; Sarjoughian, H.S. An Environment for Developing Simulatable AADL-DEVS Models. *Simul. Model. Pract. Theory* **2023**, *123*, 102690. [CrossRef]
22. Chow, A.C.; Zeigler, B.P.; Kim, D.H. Abstract simulator for the parallel DEVS formalism. In Proceedings of the Fifth Annual Conference on AI, and Planning in High Autonomy Systems, Gainesville, FL, USA, 7–9 December 1994; IEEE: Piscataway, NJ, USA, 1994; pp. 157–163.
23. Zeigler, B.P.; Praehofer, H.; Kim, T.G. *Theory of Modelling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*; Academic Press: San Diego, CA, USA, 2000.
24. Risco-Martín, J.L.; Mittal, S.; Henares, K.; Cardenas, R.; Arroba, P. xDEVS: A toolkit for interoperable modeling and simulation of formal discrete event systems. *Softw. Pract. Exp.* **2023**, *53*, 748–789. [CrossRef]
25. Popovici, K.; Mosterman, P.J. *Real-Time Simulation Technologies: Principles, Methodologies, and Applications*; CRC Press: Boca Raton, FL, USA, 2017.
26. Fernández Sebastián, O. Diseño e Implementación de un Motor de Simulación en Tiempo real Basado en el Formalismo DEVS. Bachelor's Thesis, Universidad Politécnica de Madrid, Madrid, Spain, 2023.
27. DSI. EE Modeling System. 2022. Available online: https://www.eemodelingsystem.com/user-center/downloads (accessed on 12 March 2024).
28. Sinha, S.; Goyal, N.K.; Mall, R. Reliability and availability prediction of embedded systems based on environment modeling and simulation. *Simul. Model. Pract. Theory* **2021**, *108*, 102246. [CrossRef]
29. Losada, M.; Cortés, A.; Irizar, A.; Cejudo, J.; Perez, A. A Flexible Fog Computing Design for Low-Power Consumption and Low Latency Applications. *Electronics* **2020**, *10*, 57. [CrossRef]
30. Seo, S.H.; Choi, J.I.; Song, J. Secure Utilization of Beacons and UAVs in Emergency Response Systems for Building Fire Hazard. *Sensors* **2017**, *17*, 2200. [CrossRef] [PubMed]
31. Aziz, H.; Ridley, M. Adaptive polling for responsive web applications. In Proceedings of the Information science and applications (ICISA), Krabi, Thailand, 27–29 October 2016; Springer: Singapore, 2016; pp. 1157–1167.
32. Van Mierlo, S.; Van Tendeloo, Y.; Vangheluwe, H. Debugging parallel DEVS. *Simulation* **2017**, *93*, 285–306. [CrossRef]
33. Si, H.; Aung, Z.M. Position data acquisition from NMEA protocol of global positioning system. *Int. J. Comput. Electr. Eng.* **2011**, *3*, 353. [CrossRef]
34. Wu, D.; Liu, J.; Cordova, M.; Hellevik, C.C.; Cyvin, J.B.; Pinto, A.; Hameed, I.A.; Pedrini, H.; da Silva Torres, R.; Fet, A.M. The PlastOPol system for marine litter monitoring by citizen scientists. *Environ. Model. Softw.* **2023**, *169*, 105784. [CrossRef]
35. Emergency Management Service—EFFIS Current Situation. Available online: https://effis.jrc.ec.europa.eu/apps/effis_current_situation/ (accessed on 12 March 2024).
36. GeoSphere Austria—Meteoalarm. Available online: https://www.meteoalarm.org/en/live/ (accessed on 12 March 2024).
37. European Seismic Risk Index Viewer—EFEHR. Available online: https://maps.eu-risk.eucentre.it/map/european-seismic-risk-index-viewer/ (accessed on 12 March 2024).
38. European Atlas of the Seas—European Commission. Available online: https://ec.europa.eu/maritimeaffairs/atlas/maritime_atlas (accessed on 12 March 2024).
39. Panduman, Y.Y.F.; Funabiki, N.; Fajrianti, E.D.; Fang, S.; Sukaridhoto, S. A Survey of AI Techniques in IoT Applications with Use Case Investigations in the Smart Environmental Monitoring and Analytics in Real-Time IoT Platform. *Information* **2024**, *15*, 153. [CrossRef]
40. Chacón, J.; Andrade, G.A.; Risco-Martín, J.L.; Esteban, S. A Bleeding Edge Web Application for Early Detection of Cyanobacterial Blooms. *Electronics* **2024**, *13*, 942. [CrossRef]