A model-based framework for IoT systems in wastewater treatment plants

Iván Alfonso[*β], Abel Gómez[*], Silvia Doñate[§], Kelly Garcés[†], Harold Castro[†], and Jordi Cabot[α]
[*]Internet Interdisciplinary Institute, Universitat Oberta de Catalunya, Spain
[β]Universidad Pedagógica y Tecnológica de Colombia, Colombia
[†]Department of Systems and Computing Engineering, Universidad de los Andes, Colombia
[§]Depuración de Aguas del Mediterráneo, Spain
[α]Luxembourg Institute of Science and Technology, Luxembourg

ABSTRACT Wastewater treatment is considered an essential industrial activity to protect the environment and human health. Water quality monitoring and management in a wastewater treatment plant (WWTP) can be supported by decentralized Internet of Things (IoT) systems (i.e., multi-layered systems that exploit edge and fog computing) deployed at the plant. However, the design and management of these systems requires technologies and strategies to cover tasks such as plant and IoT system modeling, deployment, and operation. In this paper, we propose an approach that includes a domain-specific language (DSL) for the specification of the process block diagram of the WWTP and the IoT system involved, a code generator that produces YAML manifests for deployment and configuration of the modeled IoT system, and a MAPE-K loop-based framework to operate and monitor the WWTP at run-time. As an example, we have modeled the process block diagram of the *Font de la Pedra* WWTP located in Spain.

KEYWORDS Wastewater Treatment Plant, Edge Computing, Domain-specific Language, Internet of Things.

1. Introduction

Water scarcity has been and will continue to be a global-scale environmental issue (Brusseau et al. 2019). This concern has motivated strategies and solutions for water reuse/recycling to treat and recover water from various sources for beneficial purposes such as agriculture, water supply, groundwater replenishment, industrial processes, and environmental restoration.

Wastewater treatment plants (WWTPs) are one of the solutions for cleaning wastewater (urban or industrial) so that it can be safely returned to the environment. Various physicochemical and biological processes and treatments must be performed on the wastewater to remove contaminants and suspended solids, break down organic matter, and disinfect the liquid. Currently, WWTPs use cyber-physical systems (CPSs) and the Internet of things (IoT) to monitor, control, and automate different processes in the plants. Devices such as sensors (e.g., level, temperature, or pH sensors), and actuators (e.g., motors, valves, or alarms) are deployed in different water treatment processes to collect real-time data and control operations. Additionally, multi-layer IoT systems that leverage edge and fog computing also deploy nodes or compute units (located in the plant) to run lightweight applications and cloud servers for the deployment and execution of the resource-intensive applications. An example could be the collection and comparison of data coming from different plants.

Although several unit operations (such as biological reactors, decanters, thickeners, etc.) are involved in wastewater treatment, the whole plant should be considered as an integrated system to study improvements in processes such as sludge control, primary sedimentation effects, and energy and nutrient recovery (Jeppsson et al. 2013). Therefore, WWTPs can be represented around a single process block diagram that illustrates the treatments, stations, operation units, and flow of water and sludge.

JOT reference format:
Iván Alfonso, Abel Gómez, Silvia Doñate, Kelly Garcés, Harold Castro, and Jordi Cabot. *A model-based framework for IoT systems in wastewater treatment plants*. Journal of Object Technology. Vol. 22, No. 2, 2023.
Licensed under Attribution 4.0 International (CC BY 4.0)
http://dx.doi.org/10.5381/jot.2023.22.2.a10

There are several ways to design the process block diagrams using multiple concepts, shapes, connectors, and notations such as those used in the diagrams designed by (Solís et al. 2022; Solon et al. 2017). However, there are challenges related to the design of these block diagrams and the IoT system involved. First, there is a need for a unified language for modeling WWTP process block diagrams, including the IoT system integrated in the different wastewater treatment stages. Second, not only is the design of the IoT system a complex task, even more complex is the management and adaptation of the system to address functional requirements and quality of service (QoS) at run-time.

Consequently, we propose a modeling-based solution consisting of an extended version of our domain-specific language (DSL) presented in (Alfonso et al. 2021a), and a extension of our code generator presented in (Alfonso et al. 2023). This paper extends our previous contributions in the following aspects:

– A domain-specific language (DSL). We have added new concepts to the language to enable the modeling of the process block diagram of a WWTP including the IoT system involved in the wastewater treatment stages. We have designed and implemented new editors to enable the modeling of the plant diagram through a graphical notation.
– Well-formedness rules. We have defined well-formedness rules applied to the design of the IoT system in general and to the design of the WWTP process block diagram. For example rules to restrict the modeling of some unit operations of the block diagram.
– A code generator and MAPE-K based framework. We have extended our code generator, which produces the code for the deployment and execution of the IoT system. We have designed and included new transformation rules to generate code and support functional and adaptation rules involving WWTP concepts such as unit operations (e.g, biological reactors, filters, and decanters). Additionally, we have reused a MAPE-K based framework (Alfonso et al. 2023) to self-adapt the system and control devices—actuators—at run-time.
– Evaluation. We have validated the modeling, code generation, and rule execution by implementing our approach in a WWTP scenario. We have also conducted focus groups with experts in the wastewater treatment domain to obtain feedback on some specific aspects of our approach.

The rest of the paper is structured as follows. Section 2 presents an overview of our model-based solution. The design of our DSL for modeling the WWTP and its IoT system is introduced in Section 3. Our MAPE-K based framework for self-adapting the system and controlling the actuators is presented in Section 4. Section 5 describes our code generator and Section 6 presents the implementation of our DSL. In Section 7, we present the validation of our approach. Related work is discussed in Section 8, and Section 9 concludes the paper.

2. Approach Overview

Our model-based approach involves multiple technologies, techniques, components and tasks categorized in two stages: design time for the specification of the WWTP process block diagram and the self-adaptive IoT system, and run-time to support the operation and adaptation of the system. Figure 1 summarizes an operational view of our architecture by distinguishing design-time (left-hand side), run-time (right-hand side), and a *code generator* component linking them.

Design-time stage. At design time, the user must specify the design of the WWTP diagram, the IoT system, and the rules governing the adaptability and functionality of the system. To support the user with this task, we have designed and provided a DSL for modeling WWTP process block diagrams, multi-layer IoT architectures (including devices and nodes of the physical, edge/fog and cloud layers), container-based applications deployed on the nodes, and rules to ensure system operation.

As shown in the Figure 1, the user builds a model—using our DSL—that describes the WWTP processes and the IoT system implied. The *code generator* then transforms the model into text, producing the code for the deployment of the IoT applications and the code required to support the execution of the system at run-time (including the code for infrastructure monitoring and system management tools). Both the DSL and the *code generator* are implemented using MPS [1], a language workbench developed by JetBrains to design DSLs.

Run-time stage. In the run-time stage, the operation and self-adaptation of the IoT system is performed. To achieve this, we have designed the architecture based on the MAPE-K loop (Kephart & Chess 2003), which has been widely employed for the design of self-adaptive systems. The four stages of the MAPE-K loop enable the *Monitoring* or collection of information on the current state of the system, the *Analysis* of the collected information, the *Planning* of the list of actions or adaptations to be performed on the system, and the *Execution* of the adaptation plan.

Our approach leverages different technologies and tools for each stage of the MAPE-K loop. For example, we use Prometheus[2]—a time-series database—to store infrastructure metrics, QoS information, and the data that will be collected by the WWTP sensors. These technologies and components are described in Section 4.

3. Design-time stage: Modeling WWTPs

To better understand and analyze the characteristics of complex domains such as software systems and their application domains, the definition of models—abstractions or generalized representations of a real world system—is often used (Brambilla et al. 2017). When dealing with complex domains, models are often built using DSLs, which provide a set of abstractions and vocabulary closer to that already used by domain experts, facilitating the modeling of new systems for that domain. This is the strategy we follow to model WWTPs.

We have developed a new version of our DSL for self-adaptive IoT systems (Alfonso et al. 2021a) to enable the specification of WWTPs and the IoT systems deployed in them.

[1] https://www.jetbrains.com/mps/
[2] https://prometheus.io/

Specifically, this new version of the DSL enables the modeling of three key types of elements:

WWTP process block diagram — We enable the specification of the main operating units such as filters, grit chambers, biological reactors, and the flow—either water or sludge—between these operating units.

IoT system — This DSL enables the modeling of the IoT system (infrastructure and container-based software), including sensors, actuators, nodes (edge, fog, and cloud), physical regions, applications, and other main concepts of the system.

Adaptation and functional rules —The modeling of functional rules to cover functional requirements—e.g., triggering alarms upon detection of unusual states—and adaptation rules for self-adaptation of the IoT system—e.g., recovery from failures—are also addressed by this DSL.

Our WWTP extension affects the first point but it also touches the other two as we will see in the next subsections.

3.1. Abstract syntax

The abstract syntax is usually represented by a metamodel that defines the domain concepts and its relationships. Figure 2 shows an excerpt of the metamodel that covers the relevant concepts for the specification of the process block diagram and the IoT system—the entire metamodel can be found in the project repository (*DSL for IoT systems in WWTPs* 2022). A *WWTPRegion* may encompass a set of unit operations that together

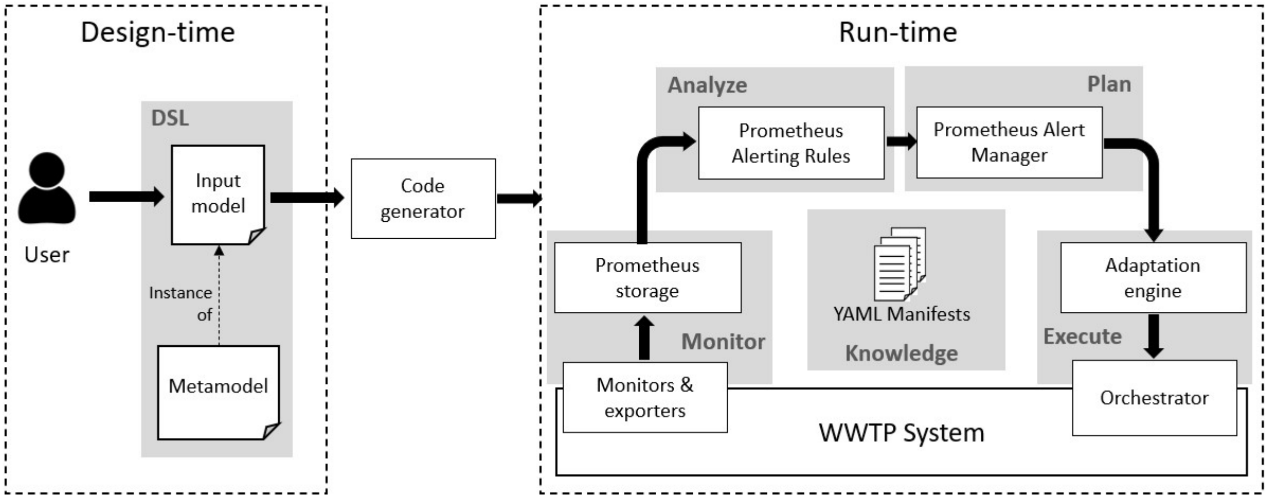


Figure 1 Solution overview

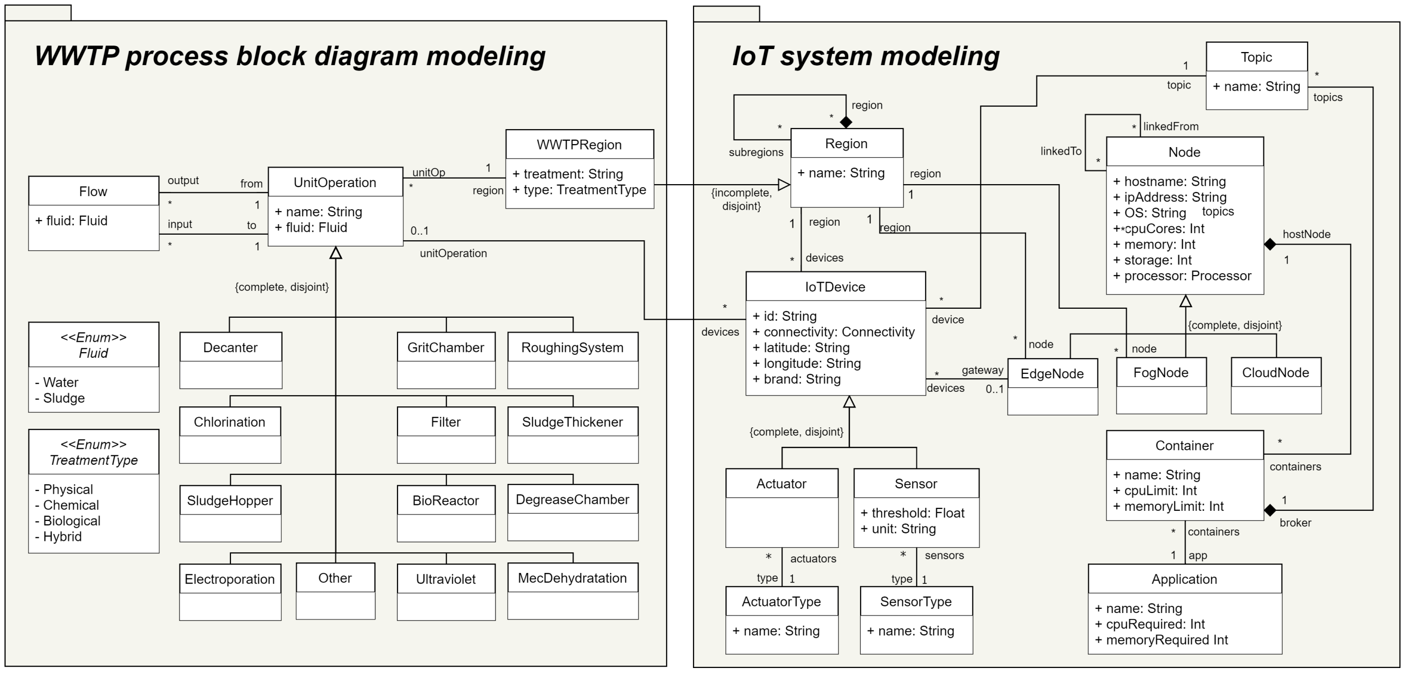


Figure 2 Metamodel of WWTP and IoT system

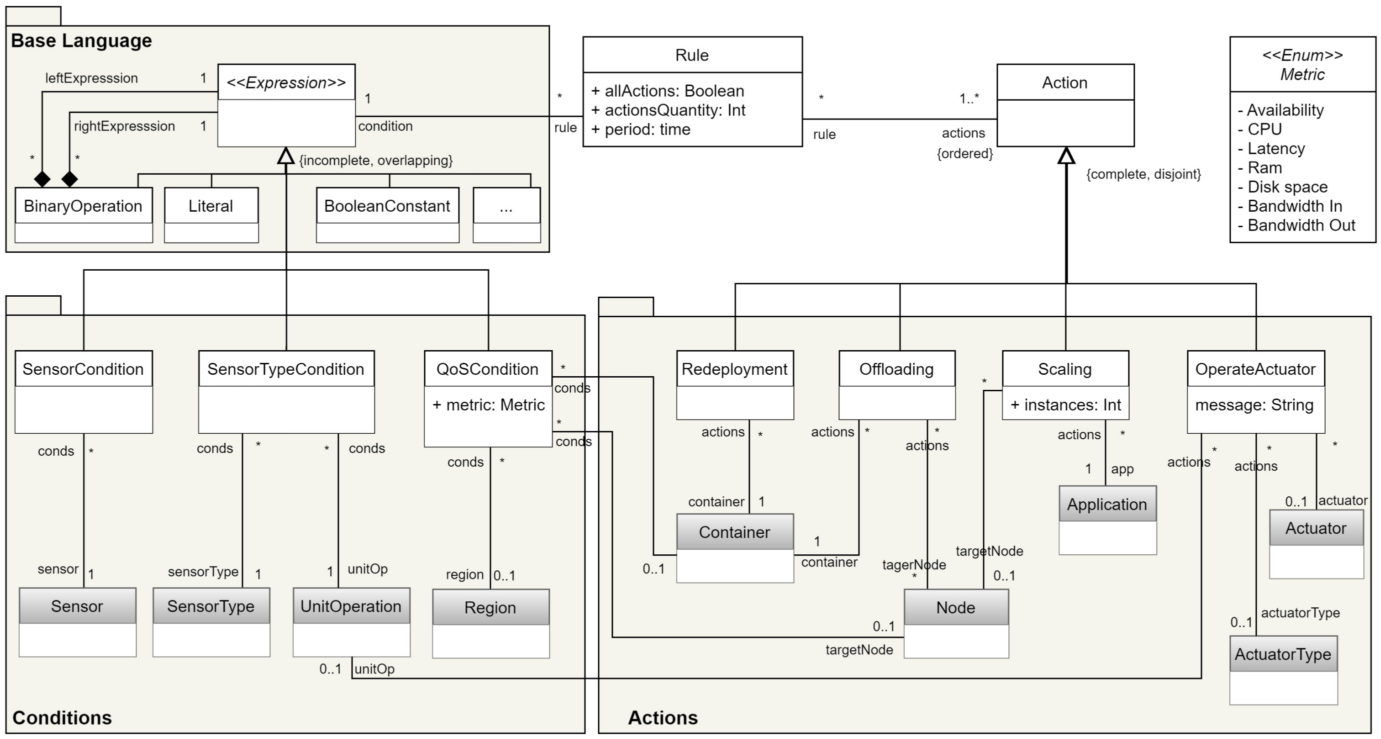


Figure 3 Rules metamodel

perform some type of wastewater treatment—such as physical, chemical, biological, or hybrid. For example, a biological treatment with activated sludge, which might involve unit operations such as decanters and biological reactors. The *UnitOperation* concept represents an operation to remove pollutants present in water and wastewater. The fluid (water or sludge) treated by the *UnitOperation* is specified by its parameter *fluid*. The different types of *UnitOperation*s are represented by the metaclasses *Decanter*, *BioReactor*, *DegreaseChamber*, and all other metaclasses that inherit from *UnitOperation*.

The *Flow* concept represents the inflows and outflows—either sludge or water—to each *UnitOperation*. An *UnitOperation* can have several inflows and outflows represented by the relationships *from* and *to*. The relationship *region* specifies the region or area in which the *UnitOperation* is physically located on the plant, and the relationship *devices* enables the specification of IoT devices, either sensors or actuators, deployed in each *UnitOperation*.

In addition to the process block diagram, this DSL addresses the modeling of the IoT system supporting the plant operation, including both infrastructure and software deployment at the nodes. Concepts such as *Sensor*, *Actuator*, *Node* (edge, fog, and cloud), *Application*, and *Container* enable the definition of the main aspects of the IoT system, including asynchronous communications (by defining *Topic*s).

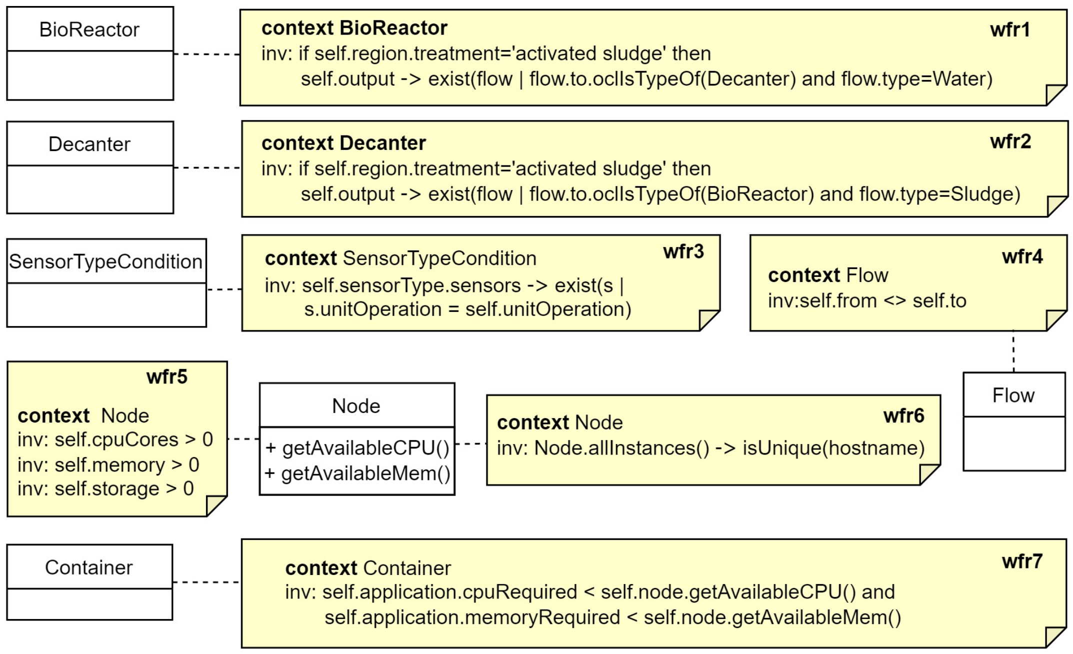
The metamodel depicted in Figure 3 defines the concepts for the specification of functional and adaptation rules.

A *Rule* is composed of an *Expression*—condition relationship—and multiple *Action*s that are executed on the system if the condition is true during a defined period—*period* attribute of *Rule*. To define the condition of a rule, we have reused and extended an existing base language, which already provides several useful concepts, such as all primitive data types and all basic arithmetic and logical operators. The metamodel extends the generic *Expression* concept by adding sensor and QoS conditions that can be combined also with all other types of expressions—e.g., *BinaryOperation*s, *Literal*s, or *BooleanConstant*s—in a complex conditional expression.

A *SensorCondition* allows defining conditions on the data collected by a specific sensor. For example to detect when some variable—such as pH, temperature, or suspended solids—exceeds a limit. Additionally, we have added the *SensorTypeCondition* concept—especially for the WWTP domain—to involve a group of sensors of a unit operation in the same condition thus avoiding the definition of a rule for each sensor. In other words, this concept is to define a condition on the data collected by a group of sensors—of the same type—of a unit operation. For example, to detect when any of the Decanter temperature sensors exceeds a limit.

Similarly, a *QoSCondition* allows detecting unusual values in QoS and system infrastructure metrics. For example, high RAM and CPU consumption in one specific node (edge, fog, or cloud), or in a group of nodes belonging to a *Region*.

With respect to *Action*s, there are four types that can be specified: (*i*) *Redeployment* consists of stopping and redeploying container instances for example when execution errors are detected that require restarting the application; (*ii*) *Offloading* task allows moving or migrating containers from a source node to a target node (usually to free up workloads on the nodes and prioritize the execution of critical system tasks); (*iii*) Horizontal *Scaling* enables automatic deployment of container-based applications on system nodes; and (*iv*) *OperateActuator* allows



```
context BioReactor                                                   wfr1
inv: if self.region.treatment='activated sludge' then
      self.output -> exist(flow | flow.to.oclIsTypeOf(Decanter) and flow.type=Water)
```

```
context Decanter                                                     wfr2
inv: if self.region.treatment='activated sludge' then
      self.output -> exist(flow | flow.to.oclIsTypeOf(BioReactor) and flow.type=Sludge)
```

```
context SensorTypeCondition                    wfr3
inv: self.sensorType.sensors -> exist(s |
      s.unitOperation = self.unitOperation)
```

```
context Flow                                   wfr4
inv:self.from <> self.to
```

```
wfr5
context  Node
inv: self.cpuCores > 0
inv: self.memory > 0
inv: self.storage > 0
```

```
context Node                                   wfr6
inv: Node.allInstances() -> isUnique(hostname)
```

```
context Container                                                    wfr7
inv: self.application.cpuRequired < self.node.getAvailableCPU() and
      self.application.memoryRequired < self.node.getAvailableMem()
```

Figure 4 Well-formedness rules

manipulating the state of system actuators by sending control commands through the corresponding actuator topic.

In the WWTP domain, it may be necessary to control groups of actuators in a unit operation. For example, closing all valves or turning on all alarms of a *Decanter* when an emergency is detected. Therefore, we have extended the *OperateActuator Action* to enable the control of a group of actuators of the same type within a unit operation. This new feature of the DSL is relevant for the case of wastewater treatment, but could also be useful in other scenarios or application cases.

To define the types of conditions, metrics to monitor, and types of actions that cover our language, we relied on our systematic literature review (SLR) (Alfonso et al. 2021b), which provides a comprehensive and holistic view of the current state of the art in IoT adaptation.

3.2. Well-Formedness Rules

Some metamodel constraints cannot be defined using only elements of the graphical metamodel syntax (Brambilla et al. 2017). An alternative to address this is to use the *Object Constraint Language* (OCL) (Cabot & Gogolla 2012), a declarative language for describing metamodel rules that are validated at the model level and known as well-formedness rules. The definition and implementation of these rules improves the accuracy of the DSL and avoids errors that could occur at run-time.

Figure 4 shows some of the well-formedness rules that we have defined using the OCL language. These rules have been expressed as invariants (*inv*), i.e., conditions that must always be true for all instances of the class defined in the *context*. These rules are the following:

Rules **wfr1** and **wfr2** restrict the unit operations *BioReactor* and *Decanter* when they conform to an activated sludge treatment: for the *BioReactor*, it must have at least one water outflow to a *Decanter*, while the *Decanter* must have at least one sludge outflow to a *BioReactor*. These two rules are designed in accordance with conventional activated sludge biological treatment processes (Gernaey et al. 2004), the most widely used biological treatment in WWTPs today (Liang et al. 2021).

The **wfr3** rule constrains that for a *SensorTypeCondition* there is at least one sensor of type *SensorType* deployed in the indicated *UnitOperation*. For example, to model a functional rule that checks the temperature in the grit chamber, there must be at least one temperature sensor in the grit chamber.

To restrict that a Flow—either water or mud—cannot have the same unit operation as source (from) and destination (to), we have defined the rule **wfr4**.

In addition to constraining the relationships of a metaclass with OCL rules, attributes can also be constrained. For instance, the rules **wfr5** ensures that the *cpuCores*, *memory*, and *storage* attributes of a node have values greater than zero, and **wfr6** guarantees that the *hostname* is unique. This type of rules were defined for all attributes of the metamodel concepts that needed to be restricted.

Finally, the **wfr7** states that a *Container* can only be defined if its host *Node* has enough available resources in terms of memory and CPU.

Note that while some rules are common in many domains—e.g., uniqueness of certain attributes—others are very specific to the domain we are metamodeling and therefore they must be provided by the domain experts.

3.3. Concrete syntax

The concrete syntax defines the notation—graphical, textual, or hybrid—for the abstract syntax. DSLs usually offer a single notation for the definition of the models. However, we take advantage of the projectional editors—Section 6 provides details

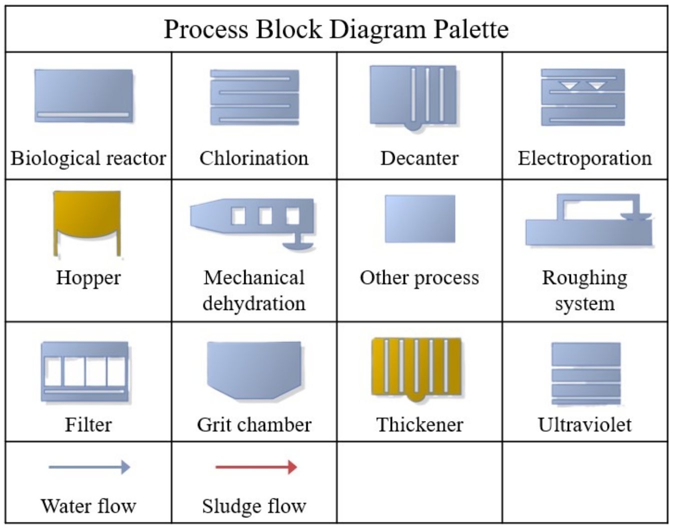


Figure 5 Process block diagram palette shapes

about these editors—to enable several notations depending on the concept to be modeled. We even configure more than one notation for some of the concepts—for example, sensors can be modeled using tabular or textual notation—so that the user can select the one that best fits his/her preferences.

Our DSL provides a graphical notation for modeling the WWTP process block diagram. We have defined fourteen shapes to instantiate the unit operations and flow types—water and sludge—covered by the metamodel. Figure 5 shows the palette of shapes that can be used to design the process block diagram of the plant. The color of the shape represents the main fluid treated by the unit operation. The blue color such as that in *Decanter* denotes water treatment, while the orange color—such as that in *Hopper* or *Thickener*—refers to sludge treatment. However, the type of fluid treated can be configured for each unit operation.

Examples of concrete notation for some of the main concepts of an IoT system for a WWTP are presented below.

Figure 6 shows the diagram—modeled using our DSL—for the *Font de la Pedra* WWTP[3], a plant located in *Muro de Alcoy*, Spain. This diagram was designed based on the model provided by our industrial partner DAM (*Depuración de Aguas del Mediterráneo*), this company manages the WWTP. This WWTP diagram—Figure 6—shows the operations of the plant which is composed of two treatment lines: (1) the water treatment line, composed of eight unit operations (blue shapes) such as *Roughing System*, *Grit Chamber*, and *Bio Reactor*; and (2) the sludge treatment line composed of three unit operations (yellow shapes) such as *Thickener* and *Hopper*. The unit operations can be instantiated by dragging and dropping the shape from the *Diagram Palette* located on the right side. Water or sludge flows can be created selecting the fluid type from the source unit operation and connecting the target unit operation.

The parameters of each unit operation defined in the block diagram can be specified via a mixed—textual and tabular—notation. For example, the specification of the *Grit Chamber* *GC-01* is shown in Figure 7, including the type of fluid processed by the unit operation, the physical region where it is located, and the list of sensors and actuators. This unit operation includes five sensors—one for pH, two for electrical conductivity, one for total suspended solids, and one for level—and three actuators—one valve and two buzzer. All sensors and actuators have a type, a brand, a connectivity type (such as ethernet, wifi, *ZigBee*, or another), and their geographic coordinates. We also cover the concepts for specifying publish/subscribe communication between IoT devices and nodes. Sensors will be publishers in topics while actuators will be subscribers. Finally, unit and threshold parameters are configurable only for devices that collect information—i.e., sensors.

The modeling of rules—functional or adaptive—is performed following a textual notation. A **condition**, a **period**, and a list of **actions** must be defined for a rule.

The **condition** can be a Boolean expression that compares two other expressions, left and right, using comparison and logical operators (such as >, <, and `&&`). Left and right expressions are enclosed by brackets.

To define a *SensorCondition*, the sensor ID, the operator, and the numerical value to be compared must be specified. For example, the condition to detect when the sensor *temp-01* exceeds 20 degrees Celsius is `(temp-01) > (20°C)`. To define a *SensorTypeCondition*, the unit operation, the sensor type, the comparison operator, and the numeric value must be specified. For example, the condition that detects when any of the temperature sensors of the unit operation *Hopper01* exceeds 50 degrees Celsius is `(Hopper01->temperature) > (50°C)`. The symbol `->` denotes the sensor type query on the unit operation. Similarly, *QoSCondition*s are defined following the same notation, specifying condition settings such as the metric, the container, the node, and the region.

The **period** of the rule is a numerical value followed by the unit of time (ms for milliseconds, s for seconds, m for minutes, h for hours, and d for days).

The **actions** of a rule are specified in a vertical list that includes the type of action and all the parameters involved in the action. When the rule is composed of more than one action, you must specify whether all actions will be executed or only a specified amount. Examples of the concrete notation of rules are presented below

Figure 8 shows two functional rules modeled for the Font de la Pedra WWTP example. The rule named *GC-level Alarm* states that if the *GC-level* sensor detects a value greater than 400 cm for 60 seconds, then the control commands *open* and *on* are sent to the *GC-valve* and *GC-alarm* actuators respectively. The second rule named *GC-ECounductivity Alarm* checks if any of the electrical conductivity sensors *EConductivity* belonging to the *GC-01 UnitOperation* detect a measurement higher than $2500\mu S/cm$ for 10 minutes, then all *Buzzer* actuators—i.e., sensors 7 and 8 in Figure 7—of the unit operation *GC-01* will be turned on.

Currently, both the values sent by the sensors and the control messages sent to the actuators are in plain text format. Therefore, sensors and actuators should support the sending and processing of that message format. Although the implementation of stan-

[3] https://www.epsar.gva.es/font-de-la-pedra

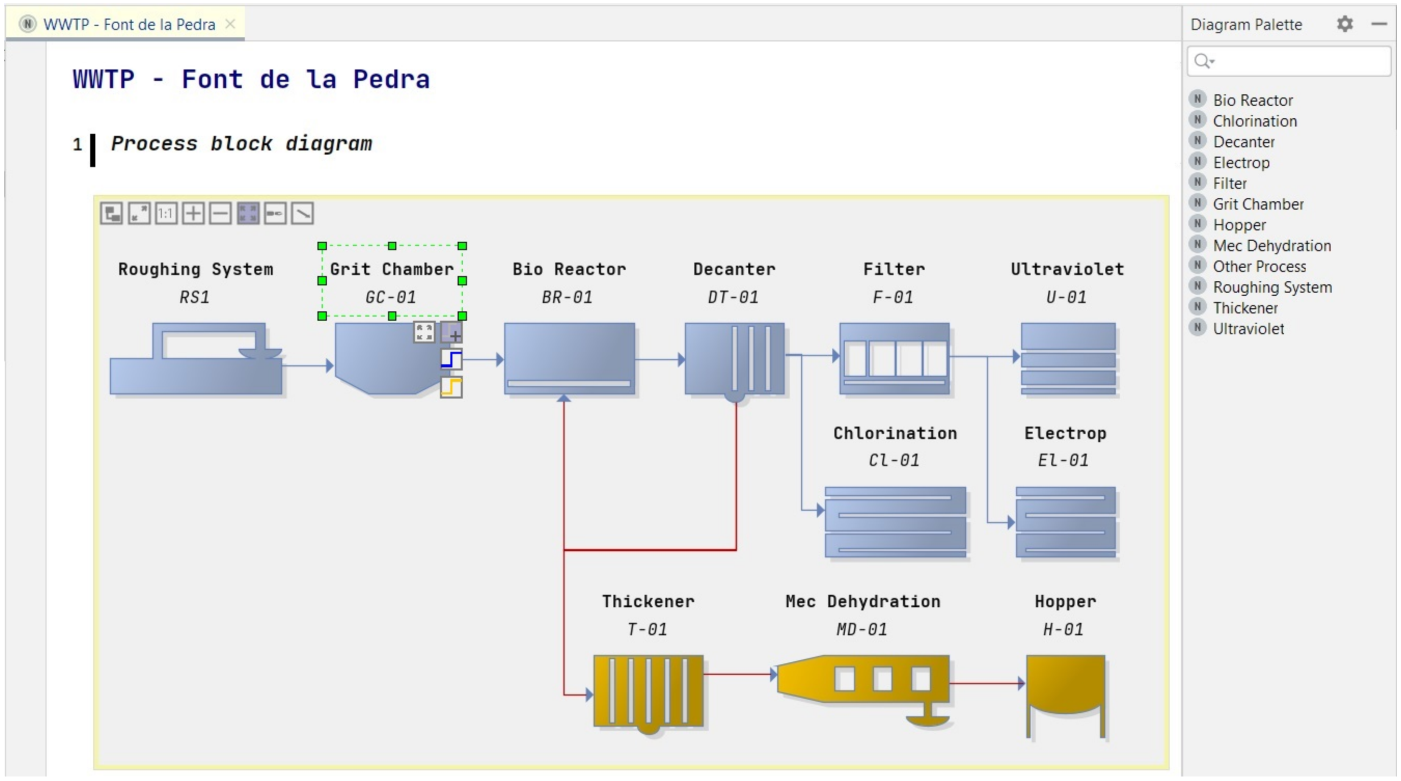
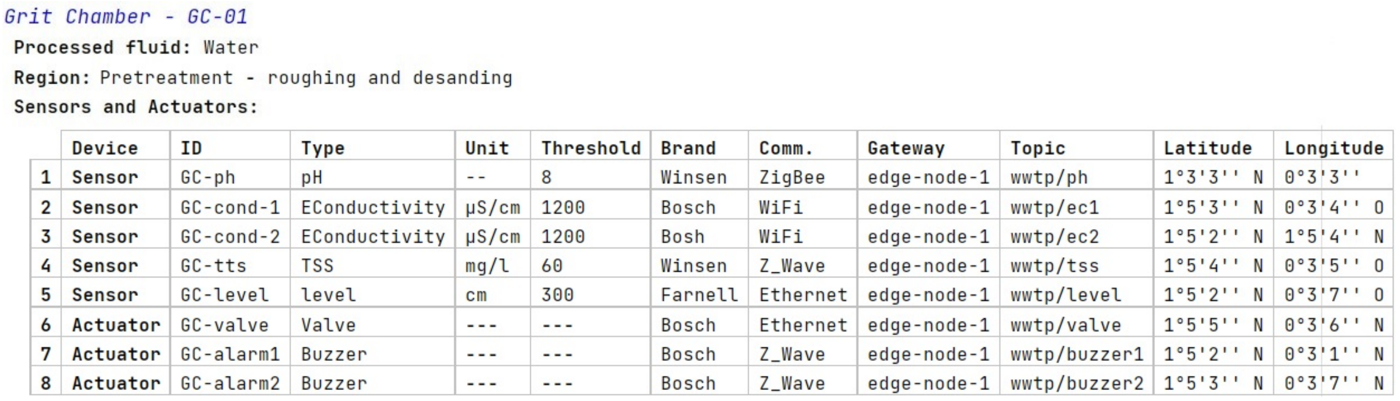


Figure 6 Process block diagram of the Font de la Pedra WWTP modeled using the DSL



Grit Chamber - GC-01

Processed fluid: Water

Region: Pretreatment - roughing and desanding

Sensors and Actuators:

	Device	ID	Type	Unit	Threshold	Brand	Comm.	Gateway	Topic	Latitude	Longitude
1	Sensor	GC-ph	pH	--	8	Winsen	ZigBee	edge-node-1	wwtp/ph	1°3'3'' N	0°3'3''
2	Sensor	GC-cond-1	EConductivity	μS/cm	1200	Bosch	WiFi	edge-node-1	wwtp/ec1	1°5'3'' N	0°3'4'' O
3	Sensor	GC-cond-2	EConductivity	μS/cm	1200	Bosh	WiFi	edge-node-1	wwtp/ec2	1°5'2'' N	1°5'4'' N
4	Sensor	GC-tts	TSS	mg/l	60	Winsen	Z_Wave	edge-node-1	wwtp/tss	1°5'4'' N	0°3'5'' O
5	Sensor	GC-level	level	cm	300	Farnell	Ethernet	edge-node-1	wwtp/level	1°5'2'' N	0°3'7'' O
6	Actuator	GC-valve	Valve	---	---	Bosch	Ethernet	edge-node-1	wwtp/valve	1°5'5'' N	0°3'6'' N
7	Actuator	GC-alarm1	Buzzer	---	---	Bosch	Z_Wave	edge-node-1	wwtp/buzzer1	1°5'2'' N	0°3'1'' N
8	Actuator	GC-alarm2	Buzzer	---	---	Bosch	Z_Wave	edge-node-1	wwtp/buzzer2	1°5'3'' N	0°3'7'' N

Figure 7 Modeling of sensors and actuators

dards such as AsyncAPI (AsyncAPI Initiative 2020) to describe message-driven APIs and improve interoperability is out of the scope of this study, we plan to address it as part of future work.

Other concepts for modeling the IoT system such as nodes, regions, brokers, applications, and containers can be specified using tabular, textual, and tree-view notations.

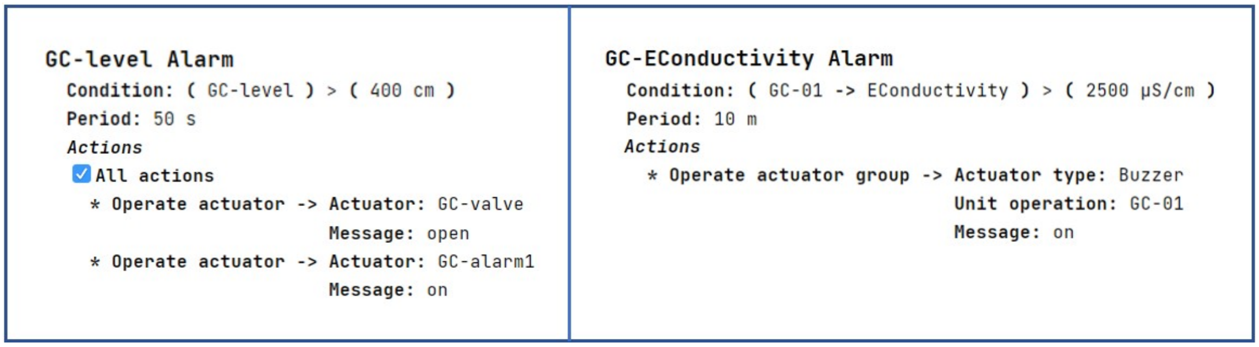
4. Run-time stage: MAPE-K based framework

The MAPE-K loop, proposed by IBM for autonomous computing, has been often employed for the design of self-adaptive systems. Indeed, MAPE-K is a reference model to implement adaptation mechanisms. This model includes four activities (monitor, analyze, plan, and execute) in an iterative feedback cycle that operate on a knowledge base. These four activities produce and exchange knowledge and information to apply adaptations and execute rules due to changes in the IoT system.

Based on the MAPE-K loop, our architecture (at run-time stage) is composed of a set of components and technologies deployed to handle the tasks in the different stages of the loop (Monitor, Analyze, Plan, and Execute).

In the **Monitor stage**, metrics on the current state of the system are collected and stored. We have categorized the metrics into two groups:

Sensor data — The data captured by the system's sensors, such as temperature, suspended solids, level, or pressure. These metrics are collected using a monitor subscribed to the topics where the sensors publish the data.



```
GC-level Alarm
  Condition: ( GC-level ) > ( 400 cm )
  Period: 50 s
  Actions
   ☑All actions
    * Operate actuator -> Actuator: GC-valve
                          Message: open
    * Operate actuator -> Actuator: GC-alarm1
                          Message: on
```

```
GC-EConductivity Alarm
  Condition: ( GC-01 -> EConductivity ) > ( 2500 µS/cm )
  Period: 10 m
  Actions
    * Operate actuator group -> Actuator type: Buzzer
                                Unit operation: GC-01
                                Message: on
```

Figure 8 Modeling example rules

Infrastructure and QoS — Infrastructure and QoS metrics (such as bandwidth, CPU usage, and availability) are collected using kube-state-metrics[4] and node-exporter[5] (container cluster monitoring tools).

Both sensor metrics and infrastructure and QoS metrics are stored in Prometheus, a time series based database. We have adopted a time-series database because, compared to other types of databases—e.g., documentary or relational databases—Prometheus is optimized to store information in a time-efficient format, enhancing the queries performed in time windows. These queries are necessary to verify the rule conditions at run-time. Additionally, Prometheus contains modules and components that facilitate the tasks performed in the later stages of our framework such as analysis and planning.

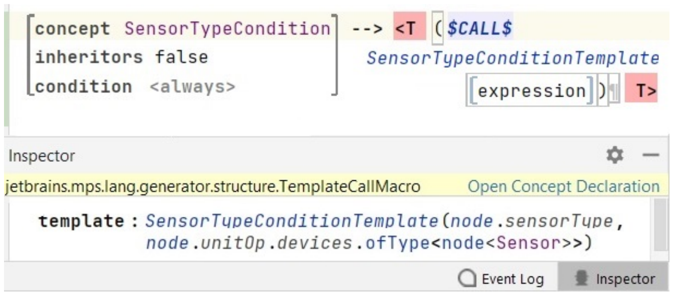
In the **Analyze stage**, the analysis tasks of the information collected in the previous stage are supported by the *Prometheus Alerting Rules* component. This tool allows defining alerting rules based on *Prometheus Query Language* (*PromQL*) that are executed periodically. We leverage the *Alerting Rules* component to define alerting rules for each functional or adaptation rule that the user has previously modeled using the DSL. When an alert is generated, i.e., when the condition of a rule is true, a notification is sent to the next stage of the MAPE-K loop.

For each notification received in the **Plan stage**, an adaptation plan is designed with the set of actions—e.g. scaling an application—to be performed on the system. To manage these notifications and organize the adaptation plans, we have used the *Prometheus Alert Manager* component, which sends the adaptation plans in JSON format via HTTP POST requests to the *Adaptation Engine*.

Finally, in the **Execute stage**, the *Adaptation Engine* receives and executes the adaptation plan through the Orchestrator. We have developed the *Adaptation Engine* using the Python language and the API to manage K3S[6], a Kubernetes-based orchestrator optimized for IoT and edge environments.

5. Code generator

In the overview architecture—cf. Figure 1—the *code generator* component is the bridge between the design stage and the run-time stage. The *code generator* performs a model-to-text (M2T) transformation taking as input the model designed by the user



```
concept SensorTypeCondition  --> <T ($CALL$
inheritors false                  SensorTypeConditionTemplate
condition <always>                [expression]) T>
```

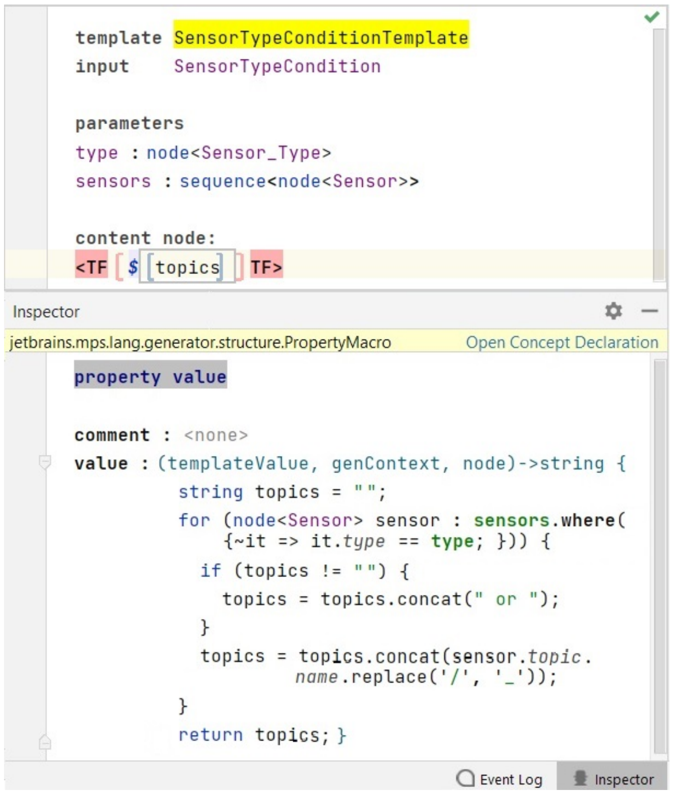
Inspector

jetbrains.mps.lang.generator.structure.TemplateCallMacro   Open Concept Declaration

```
template : SensorTypeConditionTemplate(node.sensorType,
           node.unitOp.devices.ofType<node<Sensor>>)
```

Event Log   Inspector

Figure 9 Reduction rule for the *SensorTypeCondition* concept



```
template SensorTypeConditionTemplate
input    SensorTypeCondition

parameters
type  : node<Sensor_Type>
sensors  : sequence<node<Sensor>>

content node:
<TF [ $ [topics] ] TF>
```

Inspector

jetbrains.mps.lang.generator.structure.PropertyMacro   Open Concept Declaration

```
property value

comment : <none>
value : (templateValue, genContext, node)->string {
          string topics = "";
          for (node<Sensor> sensor : sensors.where(
              {~it => it.type == type; })) {
            if (topics != "") {
              topics = topics.concat(" or ");
            }
            topics = topics.concat(sensor.topic.
                     name.replace('/', '_'));
          }
          return topics; }
```

Event Log   Inspector

Figure 10 Template to generate the PromQL expression for *SensorTypeConditions*

[4] https://github.com/kubernetes/kube-state-metrics
[5] https://github.com/prometheus/node_exporter
[6] https://k3s.io/

using the DSL. Then, several YAML[7] files are generated for the configuration and deployment of the tools and technologies used by the MAPE-K based framework. Specifically, the code for the deployment and configuration of the following artifacts is automatically generated:

- The container-based IoT applications specified in the input model.
- The monitoring tools and exporters such as kube-state-metrics, node-exporter, and mqtt-exporter.
- The *Prometheus Storage*, *Prometheus Alerting Rules*, and *Prometheus Alert Manager* components. The *PromQL* code to define the rules is also generated as a Prometheus configuration file.
- The *Adaptation Engine*.
- The *Grafana*[8] application to display the monitored data stored in the Prometheus database.

The *code generator* is implemented in MPS. The M2T transformation is performed by template-based generators, which consist of generation rules—such as root mapping, reduction, and weaving rules—and templates. To support code generation from the WWTP model (and the new concepts of or our DSL), we extended our code generator in (Alfonso et al. 2023) by developing new mapping configurations and templates. For example, to address the code generation for functional rules whose condition is *SensorTypeCondition*—i.e., rules that check a type of sensors in a unit operation—we have designed a new reduction rule and a new template. Similarly, other rules and a template were defined for the code generation of the *Operate-Actuator* action that involves a set of actuators.

5.1. Reduction rule

We have defined reduction rules for simplified transformations of the elements or nodes of the input model (i.e., the WWTP model) into text (output code), or also for executing templates that perform complex transformations. Figure 9 shows the reduction rule to generate part of the code executed in the *Analysis* stage of our approach. Specifically, this reduction rule executes the *SensorTypeConditionTemplate* for each instance of the *SensorTypeCondition* in the input model. The `CALL` macro is used to execute the template, and the Inspector box—bottom of Figure 9—shows the two parameters that are sent to the template: the sensor type of the condition, and the list of sensors of the unit operation associated.

5.2. Template

In the template object, the transformation and code generation is performed. We have implemented the *PlainText Generator* [9] plugin to define the templates of our generator. The templates contain different types of macros used to calculate the value of a property (e.g., to get the name of a container), to get the target of a reference, or to control template filling at generation time.

Figure 10 shows the definition of the *SensorTypeCondition-Template* which takes each instance of *SensorTypeCondition* as input to generate part of the *PromQL* expression of the rule condition. Basically, an iteration of a sequence of sensors—*sensors* parameter—of the same type—*type* parameter—is defined to extract the sensor topics and generate a string data with *or* logical operators.

Portions of the YAML code generated for the Prometheus Alerting Rules configuration are shown in Figure 11. This code contains the specification of a *Kubernetes ConfigMap* object with the configuration for the *Pod* running *Prometheus Alerting Rules*. The file defines the rule named *GC-EConductivity Alarm* (rule shown on the right side of the Figure 8). The *PromQL* rule expression (line 11) defines the rule condition, which generates an alert when one of the *EConductivity* sensors of the unit operation *GC-01* (i.e., the *GC-cond-1* or *GC-cond-2* sensors) detect a measure greater than 2500. The terms `wwtp_ec1` and `wwt_ec2` are the names of the *Prometheus* time series for storing sensor data. These terms correspond to the sensor topics. The *for* tag (line 12) sets the interval of the rule condition (10 minutes). Finally, adaptations label (line 14) corresponds to the actions of the rule, i.e. the sending of the control message `on` to the buzzers of the unit operation *GC-01*. An example of all the generated code can be found in the project repository (*DSL for IoT systems in WWTPs* 2022).

6. Tool support

To provide different notations (textual, graphical, and tabular) for modeling the WWTP and its IoT system, we have implemented the DSL in MPS leveraging the advantages of the projectional editors. Our prototype is open source and is available in the project repository which can be consulted at (*DSL for IoT systems in WWTPs* 2022).

Projectional editors enable editing of the model by means of projections of the abstract syntax, but the model is stored in a format (e.g. XML) independent of its concrete syntax. In other words, the user interacts with these projections, which are then translated by the editor to modify the persisted model (Völter 2011).

Defining a language in MPS involves the design of several aspects such as *Structure*, *Editor*, and *Generator*. The implementation of our DSL for IoT system modeling of a WWTP extends the aspects designed by the language presented in (Alfonso et al. 2023) as follows.

6.1. Structure

The definition of a language begins with abstract syntax. In MPS, the *Structure* aspect defines the Abstract Syntax Tree (AST) of the DSL by defining all metamodel concepts. Each concept has properties (attributes), children (composition relationships), and references. Concepts can extend from other concepts to represent inheritance relationships.

We have extended the *Structure* aspect to define new language concepts, such as those presented in the metamodel for the process block diagram and conditions and actions of a rule that involves groups of sensors and actuators. As an example

[7] YAML is a data serialization format used in this study to represent Kubernetes objects.
[8] https://grafana.com/
[9] https://jetbrains.github.io/MPS-extensions/extensions/plaintext-gen/

of a concept defined in MPS, Figure 12 shows the definition of the *SensorTypeCondition* concept, which extends *Expression*. This concept has no properties or composition relationships (*children* label), but has two references to *UnitOperation* and *Sensor_Type* with multiplicity 1.

6.2. Editor

Projectional editors define the AST code editing rules, i.e. the concrete syntax of the language. The design of the editors greatly influences the DSL usability, since these define the notation that the user will actually use to edit the model.

Editors in MPS are based on different types of cells (the smallest unit relevant for projection). Defining an editor consists of arranging cells and defining their content (Völter 2011). We have defined textual, tabular, and mixed editors by implementing the mbeddr[10] extension of MPS. This extension simplifies the definition of cells to build different types of notations. Specifically, we have defined graphical editors for each of the unit operations, tabular and textual editors to describe a unit operation including its sensors and actuators, and tree-view editors for the WWTP regions.

Figure 13 shows the graphical editor designed for the *BioReactor* concept. The concept notation is defined using a *diagram.box* (a block that can have ports, textual content, a form, and other features). In the editor section we define the textual content of the block: first the *alias* property will be displayed (such as Bio Reactor, Decanter, or Filter), and below it the name of the unit operation (a string specified by the user). The shape section is to define the figure or icon to represent the unit operation. To define these shapes, we have used the *Shapes DSL* provided by MPS, which enables the definition of shapes using Java 2D classes such as arcs, rectangles, lines, and areas.

6.3. Constraints and Type-system

Constraints aspect is to restrict the relationships between nodes as well as the allowed values for properties—e.g., to constrain the name of the unit operation to be unique. Similarly, the type-system aspect allows to provide rules to check the model code. The MPS type-system engine will evaluate the rules on-the-fly, calculate types for nodes, and report errors. We have used *Constraints* and *Type-system* aspects to define the well-formedness rules of the DSL, such as the rules shown in Figure 4.

[10] http://mbeddr.com/

7. Validation

In this section, we present first an internal validation of our approach by evaluating components such as the DSL, the code generator and the adaptation engine. We then introduce an external validation by collaborating with domain experts to obtain feedback on the usability and applicability of our approach.

7.1. Internal validation

We have performed a validation of the design-time and run-time stages of our framework by implementing a WWTP scenario to validate the modeling, code generation, and rules execution.

7.1.1. Design-time We first validated that our DSL was expressive enough to model a variety of WWTP plan designs. The DSL itself was created to be able to model the process block diagram of the Font de la Pedra WWTP (shown in Figure 6) but we then applied it to other WWTP plants such as the Algemesí WWTP [11], the Cullera WWTP [12], and even the general system-level plant part of the diagrams studied in (Solís et al. 2022; Solon et al. 2017; Márquez et al. 2022) for additional validation.

The modeled scenarios involved a variety of DSL constructs. For instance, for the IoT system of Font de la Pedra WWTP, we modeled sensors, actuators, and nodes that host applications such as the framework's components. Currently, a multiparameter probe—a device with several types of sensors—is operated in the input of the *Grit Chamber* at the *Font de la Pedra* WWTP to collect water quality data. Five variables are monitored: *total suspended solids* (TSS), *chemical oxygen demand* (COD), *electrical conductivity*, *pH*, and *temperature*. Sensors to monitor these variables were specified in the model as well as other

[11] https://www.epsar.gva.es/algemesi-albalat
[12] https://www.epsar.gva.es/index.php/cullera-0

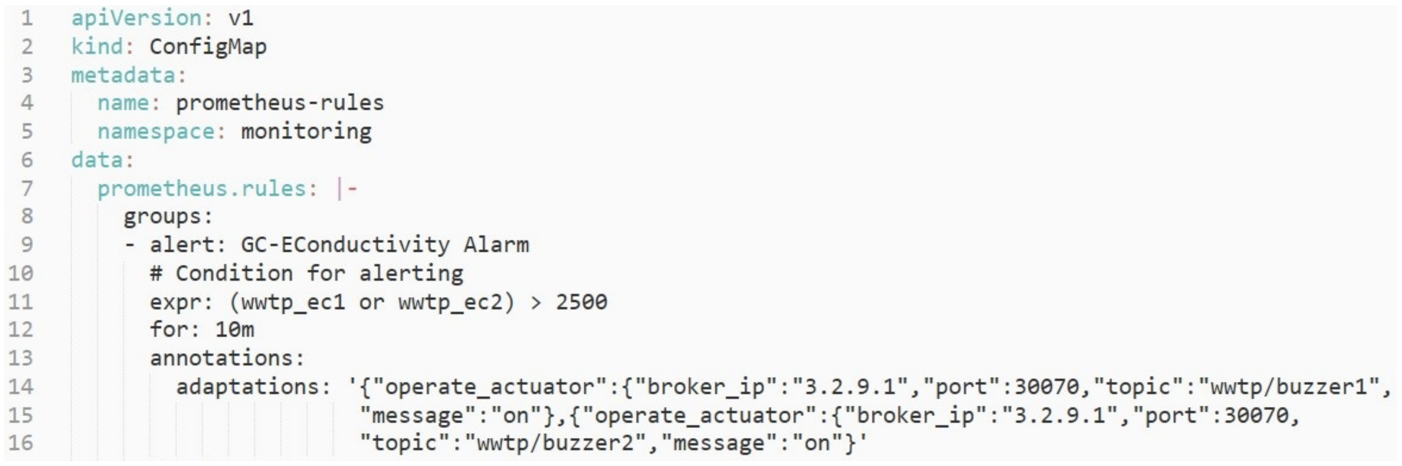
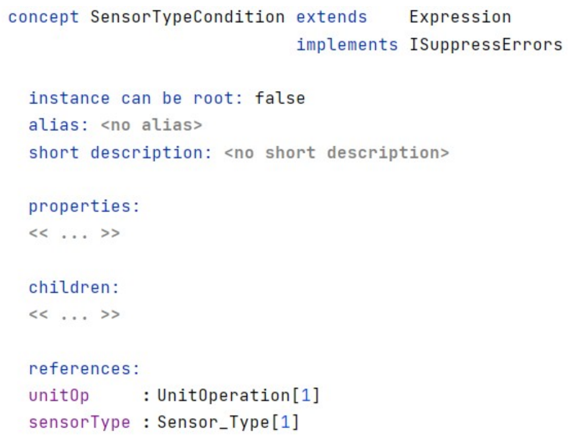
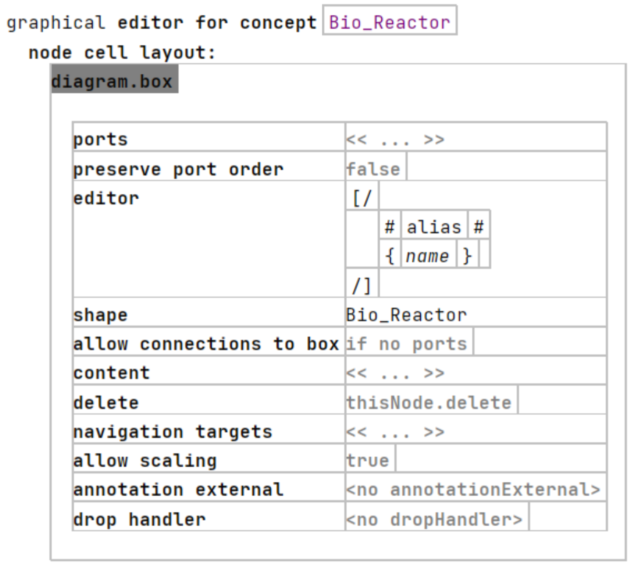

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-rules
  namespace: monitoring
data:
  prometheus.rules: |-
    groups:
    - alert: GC-EConductivity Alarm
      # Condition for alerting
      expr: (wwtp_ec1 or wwtp_ec2) > 2500
      for: 10m
      annotations:
        adaptations: '{"operate_actuator":{"broker_ip":"3.2.9.1","port":30070,"topic":"wwtp/buzzer1",
                      "message":"on"},{"operate_actuator":{"broker_ip":"3.2.9.1","port":30070,
                      "topic":"wwtp/buzzer2","message":"on"}'
```

Figure 11 Generated code for Prometheus Alerting Rules configuration



```
concept SensorTypeCondition extends    Expression
                            implements ISuppressErrors

  instance can be root: false
  alias: <no alias>
  short description: <no short description>

  properties:
  << ... >>

  children:
  << ... >>

  references:
  unitOp     : UnitOperation[1]
  sensorType : Sensor_Type[1]
```

Figure 12 *SensorTypeCondition* concept definition in MPS



```
graphical editor for concept Bio_Reactor
  node cell layout:
    diagram.box
```

ports	<< ... >>
preserve port order	false
editor	[/ # alias # { *name* } /]
shape	Bio_Reactor
allow connections to box	if no ports
content	<< ... >>
delete	thisNode.delete
navigation targets	<< ... >>
allow scaling	true
annotation external	<no annotationExternal>
drop handler	<no dropHandler>

Figure 13 Graphical editor for the *BioReactor* concept

concepts such as WWTP regions, an MQTT[13] broker, topics, and rules.

We also modeled multiple rules involving the variables monitored by the system sensors to detect when these exceed normal operational values. For example, rules to detect when the pH was not in the 6-9 range, since the microbes used in biological treatment could be killed by high acidic wastewater (Meenakshipriya et al. 2008). These and other types of rules were modeled for this test scenario.

7.1.2. Run-time To validate the run-time infrastructure we conducted some simulations with real data provided by our industrial partner.

After producing the YAML files using the code generator, we deployed our MAPE-K framework on a two-node cluster—two EC2 instances provisioned in AWS—orchestrated by K3S. The MQTT broker were deployed in the first node, and the MAPE-K framework on the other node.

[13] https://mqtt.org/

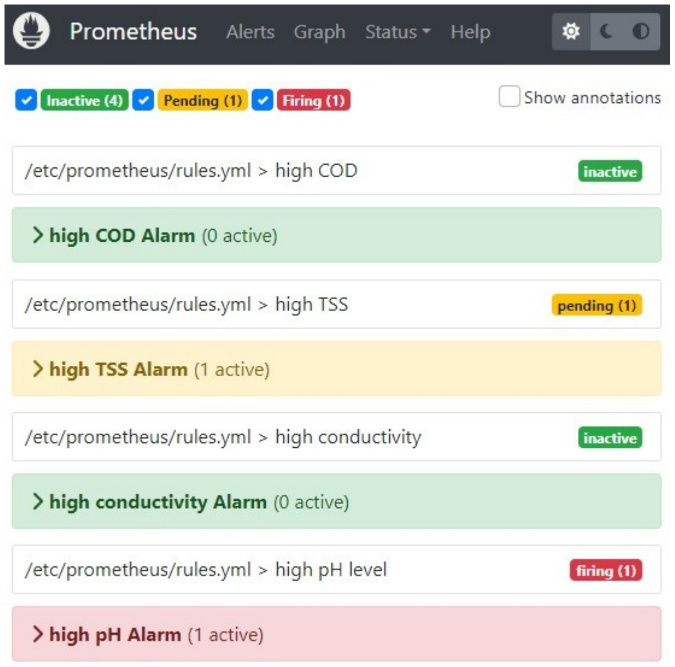
Our industrial partner provided us with a compilation of the data collected over nine months by the multiparameter probe deployed at the WWTP. The sensors were configured to monitor and record a sample every two minutes—about 390 000 data were collected per sensor. We developed a python script to read and publish this data to the broker topics simulating the behavior of the real sensors. Using the *Grafana* and *Prometheus* user interfaces (UIs), we validated the triggering of the alert rules each time the measurement threshold of a variable was exceeded during the configured period.

A rule can have three states: inactive (the rule condition is false), pending (the rule condition is true but has not exceeded the period), and firing (the rule condition is true during the period). Figure 14 shows the status of the rules configured for the WWTP. One rule was firing, one was pending, and four were inactive. The pH value increased for several minutes generating the activation—firing state—of the alarm *high pH Alarm*, while the monitored TSS level started to exceed the threshold but not yet during the period established by the rule—pending state.

7.2. External validation: focus group

Based on the recommendations for the design and application of focus groups as an empirical method for software engineering (Kontio et al. 2008), we have designed and conducted a focus group to validate our approach with experts in the wastewater treatment domain. The focus group method is a cost-effective technique to obtain insights and feedback through group interaction or discussion on a topic determined by the researcher (Morgan 1996). The design and setup of this method consists of four stages: defining the problem, planning the focus group, conducting the focus group session, and analyzing results.

Defining the problem. As a focus group is usually conducted in a short session—maximum 2 or 3 hours—, this



/etc/prometheus/rules.yml > high COD — inactive
> high COD Alarm (0 active)
/etc/prometheus/rules.yml > high TSS — pending (1)
> high TSS Alarm (1 active)
/etc/prometheus/rules.yml > high conductivity — inactive
> high conductivity Alarm (0 active)
/etc/prometheus/rules.yml > high pH level — firing (1)
> high pH Alarm (1 active)

Figure 14 Status of modeled rules - Prometheus UI

method is recommended for discussing specific aspects of our approach that can be easily analyzed by experts in the WWTP domain as part of the focus group session. Concretely, we aimed to obtain feedback on the appropriateness of the modeling of the WWTP process block diagram, specification of functional rules (e.g., specification of an alarm when high TSS is detected), and visualizations of the data using the Grafana UI.

Planning the focus group. Planning the session consists of defining a structure and schedule to cover all the topics properly during the meeting. In this case, the focus group consisted of five participants, including both authors of this paper and external experts. Specifically, the focus group was composed of two experts in model-driven technologies who managed the session, and three experts in WWTP wastewater treatment processes who work at DAM[14]. We designed a presentation to introduce the objective of the focus group and describe our approach. We also prepared a demonstration of the use of the DSL, the code generation, the deployment of the system, and the visualization of the data collected by the sensors and the status of the rules in Grafana app. Finally, we carefully structured and defined 12 questions to discuss with the participants about the modeling of the plant block diagram, modeling of the functional rules, and visualization of data and rules in Grafana. Both the presentation and the questions can be consulted in the project repository [15].

Conducting the focus group session. The session was conducted in virtual mode for an hour and a half and was recorded for later analysis. Alternately, one co-author of the paper took notes during the discussion, while another co-author worked as a facilitator of the session by motivating the participants to discuss and by leading the discussion. The discussions were semi-structured, which means that we had carefully defined the question areas, but not all the single questions in detail and we were able to integrate new questions depending on the evolution of the conversation.

Analyzing results. We have analyzed the information collected and the recorded audio of the discussion. Below we summarize and discuss the perspectives, feedback, benefits, and suggestions of improvement mentioned by the focus group.

- The modeling of the process block diagram of the plant allows to represent graphically the different biological, chemical, hybrid, or physical treatment processes. The nomenclature is clear and the procedure to specify the model is relatively simple. Some parameters could be added in the specification of each operation unit to enrich the model. For example, information on water or sludge flow rates, or the exact number of UV units in an *Ultraviolets* process.
- Two aspects were discussed to improve or add to the condition specification of a rule: (1) the definition of conditions with two or more related periods. For example, the following condition has two periods related to two different metrics: if the pH > 9 for 30 seconds and the TSS > 60 mg/l for one minute. (2) including the calculation of statistical measures such as variance or dispersion of the values of a variable. For example, to detect if in the last hour the pH value had a significant variability that could affect the health of the bacteria.
- The definition of alarms by modeling functional rules using the DSL requires less effort compared to the current system implemented in many of the WWTPs managed today by DAM. Currently, many WWTPs implement a Supervisory Control and Data Acquisition (SCADA) system, which allows configuring visual alert thresholds for some variables only if the functionality was developed with the same system. Otherwise, it would require the intervention of an expert technician in the development of the SCADA system to develop and implement a rule or visual alarm.
- Although the visualizations offered by Grafana show the sensed information and the status of the rules in real time, a feature that could be included is a visualization that shows information directly on the process block diagram—similar to SCADA systems. For example, a graph of the block diagram with the status of the rules in the different operation units.
- A functionality of interest in this domain stated by the participants and that we consider as part of future work is the development of machine learning algorithms to calculate the trend of critical variables. For example, to predict significant alterations in the pH and chemical oxygen demand levels and to avoid affecting the activity of the biomass in the biological treatment or the sedimentation properties of the biological sludge.
- Finally, an important advantage of our approach with respect to the system implemented in several WWTPs was discussed. Currently, WWTPs are automated through SCADA systems that centralize the control and monitoring of the plant in a monolithic system. To query the information collected by the plant's sensors from an external or public network, it is necessary to access the system through a virtual private network (VPN). However, accessing the plant's local network generates security risks that could be minimized by deploying multilayer architectures—as we propose—to isolate and protect critical information and functionalities.

8. Related work

The use of water quality monitoring systems in WWTPs can improve the tasks and processes performed such as the collection and visualization of near real-time information, the analyses of effluent water conditions to verify that it meets regulatory standards (Martínez et al. 2020), and the detection of alert states—e.g., spills. Water quality monitoring can be based on IoT systems through the use of embedded devices in the environment exchanging information (Perumal et al. 2015).

Indeed, the design and modeling, deployment, and management of IoT systems in multiple areas have been under research for some years. Some approaches (Moghaddam et al. 2020; Eterovic et al. 2015; Yigitoglu et al. 2017) use generic languages such as Unified Modeling Language (UML) (OMG 2017), Finite-state machine (FSM) (Brand & Zafiropulo 1983),

[14] https://www.dam-aguas.es/
[15] https://github.com/SOM-Research/WWTP-DSL/tree/main/docs/focus-group

Queuing Network (QNs) (Wu & Liu 2007), and YAML to model aspects of the IoT system such as its architecture, software deployment, or self-adaptive capabilities. However, because these solutions implement general-purpose software, it is challenging to cover all the relevant concepts of IoT architectures. Even more challenging when dealing with specific types of complex domains such as wastewater treatment that could benefit from their own DSL to capture all the rich semantics of the domain.

The use of DSLs has already been exploited to support the specification of several aspects of an IoT system. Some DSLs (Gomes et al. 2017; Pramudianto et al. 2016; García et al. 2020) have been focused on reducing application development and deployment of IoT applications at nodes and end-devices. However, these solutions are focused on the device layer and do not address the management and adaptation of the run-time system. *SimulateIoT* (Barriga et al. 2021) does cover the mutli-layered dimension of an IoT architecture and can be configured to generate notifications from the analysis of data sensors but there is no support to write rules that influence other aspects of the system like the actuators. Similarly, *CAPS* (Muccini & Sharaf 2017) addresses adaptations at the software component level without supporting device-level actions. A self-adaptive framework based on MAPE-K loop is proposed by (Lee et al. 2019), while this approach does cover the rules aspect, it does lack again the capacity to model the overall architecture and the relationships between their components at different layers.

To sum up, none of the previous approaches cover multi-layered IoT systems while at the same time comprising a rule-based language to execute system adaptations based on a variety of sensor conditions. Additionally, they all lack primitives to easily model the specifics of wastewater treatment concepts.

Specific diagrams for WWTP processes appear in works from water research and chemical engineering domains (Solís et al. 2022; Solon et al. 2017; Márquez et al. 2022). However, these works use the diagrams as drawings for illustrative purposes, none of them propose specific semantics for the symbols appearing in the diagrams. Different notations, shapes, and colors are used to represent the concepts—e.g., unit operations. We believe a language like ours could benefit this community by providing them with a unified representation of WWTP processes so that they can more easily share information among the different groups.

To the best of our knowledge, ours is the first approach that enables graphical modeling of WWTPs process block diagrams, the IoT system involved, and functional and adaptation rules executed at run-time backed with a real modeling language with concrete semantics. Our solution automatically generates code for the deployment and configuration of applications, technologies, and tools to manage the operation—triggering of rules—of the WWTP system.

9. Conclusion

IoT systems can improve the automation of industrial processes in WWTPs such as water quality monitoring. We propose a model-based approach to facilitate the design, deployment, and management of multi-layer IoT systems embedded in different treatment processes of a WWTP. To support the design and modeling of the WWTP process block diagram, the IoT system and its functional and adaptation rules, we designed and implemented a DSL using MPS. Our DSL combines several notations—textual, graphical, and mixed—for system specification. As a running example, we presented the modeling of the process block diagram of the WWTP *Font de la Pedra* and an IoT system that monitors water quality variables.

Moreover, to support the deployment and management of the WWTP IoT system, we implemented a code generator and framework based on the MAPE-K loop. Our code generator produces YAML manifests for the deployment and configuration of container-based applications and technologies that implement the framework at each stage of the MAPE-K loop. For example, the code to deploy and configure the *Prometheus* database is generated. Finally, we performed an internal validation of our approach and an external validation with a group of domain experts.

As part of future work, we plan to enrich the metamodel to include more fine-grained concepts and parameters that describe with better detail the internal specific chemical and biological processes taking place in the units such as the parameters for anaerobic digestion, ion exchange, and anaerobic treatment. This will open the door to more advanced simulations of the plant behaviour under different environmental conditions.

Additionally, we plan to address the improvements suggested by the focus group participants. For example, the generation of visualizations using the block diagram to recreate a view similar to that offered by SCADA systems. Finally, we plan to integrate our solution with message formatting standards (e.g., AsyncAPI) to replace the current communications format (plain text format) between the device layer and the upper layers.

Acknowledgments

This work has been partially funded by the Spanish government (LOCOSS project - PID2020-114615RB-I00 / AEI / 10.13039/501100011033), the Colombian government (*Becas del Bicentenario* program - Art. 45, law 1942 of 2018), and TRANSACT project - the ECSEL Joint Undertaking (JU) under grant agreement No 101007260. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Netherlands, Finland, Germany, Poland, Austria, Spain, Belgium, Denmark, Norway. In addition, Jordi Cabot is supported by the Luxembourg National Research Fund (FNR) PEARL program, grant agreement 16544475

References

Alfonso, I., Garcés, K., Castro, H., & Cabot, J. (2021a). Modeling self-adaptative IoT architectures. In *2021 ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C)* (pp. 761–766).

Alfonso, I., Garcés, K., Castro, H., & Cabot, J. (2021b). Self-adaptive architectures in IoT systems: a systematic literature review. *Journal of Internet Services and Applications*, *12*(1), 1–28.

Alfonso, I., Garcés, K., Castro, H., & Cabot, J. (2023). A model-based infrastructure for the specification and runtime execution of self-adaptive IoT architectures. *Computing*, 1–24.

AsyncAPI Initiative. (2020). *AsyncAPI specification 2.0.0.* (URL: https://www.asyncapi.com/docs/specifications/2.0.0/, last accessed May 2021)

Barriga, J. A., Clemente, P. J., Sosa-Sánchez, E., & Prieto, Á. E. (2021). SimulateIoT: Domain specific language to design, code generation and execute IoT simulation environments. *IEEE Access*, *9*, 92531–92552.

Brambilla, M., Cabot, J., & Wimmer, M. (2017). *Model-driven software engineering in practice, 2nd edn. Synthesis Lectures on Software Engineering*. USA: Morgan & Claypool Publishers.

Brand, D., & Zafiropulo, P. (1983). On communicating finite-state machines. *Journal of the ACM (JACM)*, *30*(2), 323–342.

Brusseau, M., Ramirez-Andreotta, M., Pepper, I., & Maximillian, J. (2019). Environmental impacts on human health and well-being. In *Environmental and pollution science* (pp. 477–499). Elsevier.

Cabot, J., & Gogolla, M. (2012). Object constraint language (OCL): A definitive guide. In M. Bernardo, V. Cortellessa, & A. Pierantonio (Eds.), *Formal Methods for Model-Driven Engineering - 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM* (Vol. 7320, pp. 58–90). Springer.

DSL for IoT systems in WWTPs. (2022). https://github.com/SOM-Research/WWTP-DSL.

Eterovic, T., Kaljic, E., Donko, D., Salihbegovic, A., & Ribic, S. (2015). An internet of things visual domain specific modeling language based on UML. In *2015 XXV International Conference on Information, Communication and Automation Technologies (ICAT)* (pp. 1–5).

García, C. G., Meana-Llorián, D., García-Díaz, V., Jiménez, A. C., & Anzola, J. P. (2020). Midgar: Creation of a graphic domain-specific language to generate smart objects for internet of things scenarios using model-driven engineering. *IEEE Access*, *8*, 141872–141894.

Gernaey, K. V., Van Loosdrecht, M. C., Henze, M., Lind, M., & Jørgensen, S. B. (2004). Activated sludge wastewater treatment plant modelling and simulation: state of the art. *Environmental Modelling & Software*, *19*(9), 763–783.

Gomes, T., Lopes, P., Alves, J., Mestre, P., Cabral, J., Monteiro, J. L., & Tavares, A. (2017). A modeling domain-specific language for IoT-enabled operating systems. In *IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society* (pp. 3945–3950).

Jeppsson, U., Alex, J., Batstone, D. J., Benedetti, L., Comas, J., Copp, J., … others (2013). Benchmark simulation models, quo vadis? *Water Science and Technology*, *68*(1), 1–15.

Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer*, *36*(1), 41–50.

Kontio, J., Bragge, J., & Lehtola, L. (2008). The focus group method as an empirical tool in software engineering. *Guide to advanced empirical software engineering*, 93–116.

Lee, E., Seo, Y.-D., & Kim, Y.-G. (2019). Self-adaptive framework based on MAPE loop for internet of things. *Sensors*, *19*(13), 2996.

Liang, T., Elmaadawy, K., Liu, B., Hu, J., Hou, H., & Yang, J. (2021). Anaerobic fermentation of waste activated sludge for volatile fatty acid production: recent updates of pretreatment methods and the potential effect of humic and nutrients substances. *Process Safety and Environmental Protection*, *145*, 321–339.

Márquez, P., Gutiérrez, M., Toledo, M., Alhama, J., Michán, C., & Martín, M. (2022). Activated sludge process versus rotating biological contactors in WWTPs: Evaluating the influence of operation and sludge bacterial content on their odor impact. *Process Safety and Environmental Protection*, *160*, 775–785.

Martínez, R., Vela, N., El Aatik, A., Murray, E., Roche, P., & Navarro, J. M. (2020). On the use of an IoT integrated system for water quality monitoring and management in wastewater treatment plants. *Water*, *12*(4), 1096.

Meenakshipriya, B., Saravanan, K., Shanmugam, R., Sathiyavathi, S., et al. (2008). Study of pH system in common effluent treatment plant. *Modern Applied Science*, *2*(4), 113–113.

Moghaddam, M. T., Rutten, E., Lalanda, P., & Giraud, G. (2020). Ias: an IoT architectural self-adaptation framework. In *European Conference on Software Architecture* (pp. 333–351).

Morgan, D. L. (1996). Focus groups. *Annual review of sociology*, *22*(1), 129–152.

Muccini, H., & Sharaf, M. (2017). Caps: Architecture description of situational aware cyber physical systems. In *2017 IEEE International Conference on Software Architecture (ICSA)* (pp. 211–220).

OMG. (2017). *OMG Unified Modeling Language (OMG UML), Ver. 2.5.1.* (https://www.omg.org/spec/UML/2.5.1/)

Perumal, T., Sulaiman, M. N., & Leong, C. Y. (2015). Internet of Things (IoT) enabled water monitoring system. In *2015 IEEE 4th Global Conf. on Consumer Electronics* (pp. 86–87).

Pramudianto, F., Eisenhauer, M., Kamienski, C. A., Sadok, D., & Souto, E. J. (2016). Connecting the internet of things rapidly through a model driven approach. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)* (pp. 135–140).

Solís, B., Guisasola, A., Flores-Alsina, X., Jeppsson, U., & Baeza, J. A. (2022). A plant-wide model describing GHG emissions and nutrient recovery options for water resource recovery facilities. *Water research*, *215*, 118223.

Solon, K., Flores-Alsina, X., Mbamba, C. K., Ikumi, D., Volcke, E., Vaneeckhaute, C., … others (2017). Plant-wide modelling of phosphorus transformations in wastewater treatment systems: Impacts of control and operational strategies. *Water Research*, *113*, 97–110.

Völter, M. (2011). Language and IDE modularization, extension and composition with MPS. *Pre-proceedings of Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE)*, 395–431.

Wu, C., & Liu, Y. (2007). Queuing network modeling of driver workload and performance. *IEEE Transactions on Intelligent Transportation Systems*, *8*(3), 528–537.

Yigitoglu, E., Mohamed, M., Liu, L., & Ludwig, H. (2017). Foggy: a framework for continuous automated IoT application deployment in fog computing. In *2017 IEEE Inter-*

national Conference on AI & Mobile Services (AIMS) (pp. 38–45).

About the authors

Iván Alfonso is researcher of the *Internet Interdisciplinary Institute*, a research center of the *Universitat Oberta de Catalunya* (UOC), Spain. In 2022, he got his PhD degree in Engineering from the Universidad de los Andes and his PhD degree in Network and Information Technologies from the UOC. His research interest focus mainly on the fileds of Software Engineering, Model-Driven Engineering, and Internet of Things. You can contact the author at ialfonsod@uoc.edu.

Abel Gómez is a senior researcher of the *Internet Interdisciplinary Institute*, a research center of the *Universitat Oberta de Catalunya*, Spain. Previously, he has hold different positions at the *Universidad de Zaragoza*, the *École des Mines de Nantes & Inria*, and the *Universitat Politècnica de València*; being this latter institution where he obtained his PhD degree in Computer Science.
His research interests fall in the broad field of Model-Driven Engineering (MDE), and his research lines have evolved in two complementary directions: on the one hand, the development of core technologies to support MDE activities; and on the other hand, the application of MDE techniques to solve Software Engineering problems. You can contact the author at agomezlla@uoc.edu or visit https://abel.gomez.llana.me.

Silvia Doñate is an agricultural engineer specialized in wastewater treatment and innovative procedures for the management and supervision of facilities. As head of the Innovation Department of the company *Depuración de Aguas del Mediterráneo*, her research in environmental engineering of the water cycle has been published in specialized journals and congresses. You can contact the author at silvia.donate@dam-aguas.es.

Kelly Garcés is associate professor of the *Department of Systems and Computing Engineering* at the *Universidad de los Andes* (Bogota, Colombia). Prior to this, she was an R&D engineer and software engineer at *Netfective Technology*. She received her Ph.D. in September 2010 from the *Université de Nantes*. In 2011, she was a postdoctoral fellow at the *INRIA laboratory*. She has participated in research and development projects (proprietary or open source) since 2005, financed with private and public funds. Her research interests are software architecture, evolution and maintenance of complex software systems (e.g. legacy applications, IoT systems, web transactional systems, microservice-based applications, etc.). Some of her lectures are: Software Architecture and Design, Software for Internet of Things (IoT), Software Modeling, Model Driven Engineering. You can contact the author at kj.garces971@uniandes.edu.co.

Harold Castro is professor and researcher in distributed systems with over 20 years of experience in the HPC field, from massively parallel systems to current cloud computing technologies, including cluster and grid computing. Professor Castro has been the principal investigator of several projects and a technical advisor in helping organizations leverage innovative technologies related to their research area. He is also the author of many publications in books, journals, and national and international conferences. He has also worked on highly complex research projects and consulting in various local and international organizations. You can contact the author at hcastro@uniandes.edu.co.

Jordi Cabot is an FNR Pearl Chair and the head of the Software Engineering RDI Unit at the *Luxembourg Institute of Science and Technology*. His research interests include software modeling and low-code technologies, pragmatic formal model verification, analysis of open source/open data communities and the role AI can play in software development (and vice versa). You can contact the author at jordi.cabot@list.lu or visit https://jordicabot.com.